महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त) (ISO 21001:2018) (ISO/IEC 27001:2013)

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका

(Learning Material)

SOFTWARE ENGINEERING

(315323)

K Scheme

सॉफ्टवेअर इंजिनिअरींग

संगणक अभियांत्रिकी गट
(के स्कीम)

मराठी–इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील पाचवे सत्र पदविका)

शिक्षण पुस्तिका

(Learning Material)

सॉफ्टवेअर इंजिनिअरिंग

Software Engineering

(315323)

संगणक अभियांत्रिकी गट

(के-स्कीम)

K-Scheme

मराठी-इंग्रजी (द्विभाषिक) माध्यम

(अभियांत्रिकी व तंत्रज्ञानातील पाचवे सत्र पदविका)

महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त)(ISO 21001:2018) (ISO/IEC 27001:2013)

सॉफ्टवेअर इंजिनिअरिंग
Software Engineering
(315323)

मार्गदर्शक

प्रा. विजय नामदेवराव कुकरे
विभागप्रमुख, संगणक अभियांत्रिकी

प्रमुख समन्वयक

प्रा. गजानन रेवणसिद्ध धरणे
प्राचार्य

प्रकल्प समन्वयक

डॉ. नीता संगप्पा आळंगे
विभागप्रमुख, माहिती व तंत्रज्ञान

संकलक

डॉ. नीता संगप्पा आळंगे
विभागप्रमुख, माहिती व तंत्रज्ञान

प्रा. शीतल चन्नप्पा सावळगी
अधिव्याख्याता, माहिती व तंत्रज्ञान

प्रा. सागर सोमय्या हिरेमठ
अधिव्याख्याता, माहिती व तंत्रज्ञान

प्रा. रक्षा दिलीप मुथा
अधिव्याख्याता, माहिती व तंत्रज्ञान

प्रा. श्रुतिका संदिप तळेकर
अधिव्याख्याता, संगणक अभियांत्रिकी

महाराष्ट्र राज्य तंत्र शिक्षण मंडळ.

(स्वायत्त) (ISO: २१००१:२०१८) (ISO/IEC: २७००१-२०१३)

शासकीय तंत्रनिकेतन इमारत, चौथा मजला, ४९, खेरवाडी, बांद्रा (पूर्व), मुंबई - ४०० ०५१.

दूरध्वनी क्र.: ०२२-६२५४२१०० / १५३ / १७०

email : director@msbte.com web site : www.msbte.ac.in

प्रास्ताविक

महाराष्ट्र राज्यातील पदविका स्तरावरील तंत्रशिक्षणामध्ये विद्यार्थ्यांचे रोजगार कौशल्य विकसित करून विद्यार्थ्यांचा सर्वांगीण विकास घडवून आणण्याकरिता महाराष्ट्र राज्य तंत्रशिक्षण मंडळ कटिबद्ध आहे. उद्योगधंद्यातील बदलत्या तंत्रज्ञानाशी संबंधित गरजा लक्षात घेऊन महाराष्ट्र राज्य तंत्र शिक्षण मंडळाकडून पदविका अभ्यासक्रम वेळोवेळी अद्यावत करण्यात येतो. अभियांत्रिकी पदविका अभ्यासक्रम शिकत असताना संकल्पनात्मक ज्ञान, सुसंगत संदर्भ, प्रश्न विचारणे, विश्वसनीय पुरावे, कारणमीमांसा आणि सुस्पष्ट निकष यांचा वापर करून अर्थाची उकल करण्याची, विश्लेषण व मूल्यमापन करण्याची तसेच तर्काने अनुमान काढण्याची क्षमता म्हणजेच चिकित्सक विचार विद्यार्थ्यांमध्ये अधिक दृढ होतील असा मला विश्वास आहे. जेव्हा विद्यार्थी ज्ञान मिळवण्याच्या माध्यमाशी पूर्णपणे परिचित आणि सोयीस्कर असतात, तेव्हा त्यांच्यासाठी वर्गातील चर्चेत भाग घेणे सोपे होते, संकल्पनात्मक व सैद्धांतिक बाबींचे आकलन परिपूर्ण होते, संज्ञानात्मक क्षमता सुधारते आणि त्यांचा आत्मविश्वास देखील वाढतो. या सर्व गोष्टींचा विचार करून मंडळाकडून शैक्षणिक सामुग्रीची निर्मिती करण्यात आलेली आहे. भारत देश हा खेड्यापाडयातून विकसित झालेला देश असून ग्रामीण भागातील विद्यार्थ्यांना तांत्रिक शिक्षण घेताना भाषेचा अडसर न येता तांत्रिक बाबींचा आशय समजून घेणे शक्य होईल या दृष्टिकोनातून महाराष्ट्र राज्य तंत्र शिक्षण मंडळाने पदविका स्तरावरील तांत्रिक शिक्षणाकरीता विद्यार्थ्यांना मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय उपलब्ध करुन दिलेला आहे.

राष्ट्रीय शैक्षणिक धोरण-२०२० प्रादेशिक भाषेतील शिक्षणास प्रोत्साहन देते, ज्यामुळे विद्यार्थ्यांना तांत्रिक अभ्यासक्रमांसाठी प्रादेशिक भाषेतुन शिक्षणाचे माध्यम निवडता येते. त्या अनुषंगाने प्रादेशिक भाषांमध्ये तांत्रिक सामग्री आणि अभ्यास सामग्रीचा विकास आणि भाषांतर करण्याची आवश्यकता आहे. या धोरणास अनुसरुन मंडळाने भागधारकांसाठी शैक्षणिक वर्ष २०२१-२२ पासून I-Scheme तसेच शैक्षणिक वर्ष २०२३-२४ पासून K-Scheme मध्ये व्दिभाषिक माध्यमाचा पर्याय प्रथम ते तृतीय वर्षाकरिता उपलब्ध करून दिलेला आहे. या पर्यायास अनुसरून मंडळाने मराठी-इंग्रजी द्विभाषिक शैक्षणिक सामग्रीही संबंधीत विद्यार्थी व अधिव्याख्यातांकरिता उपलब्ध करून दिली आहे.

पदविका स्तरावरील तंत्रशिक्षण अधिक दर्जेदार करण्यासाठी महाराष्ट्रातील अनुभवी व तज्ञ अध्यापकांनी व्यवहारिक मराठी भाषा व इंग्रजी भाषेतील तांत्रिक शब्दावली यांचा वापर करून मराठी इंग्रजी भाषेचा सुवर्णमध्य साधण्याचा प्रयत्न केलेला आहे. मंडळाच्या स्तरावर गठीत सुकाणू समितीमार्फत सदर शैक्षणिक सामुग्रीचा दर्जा, तसेच इतर बाबींची तपासणी करण्यात आलेली आहे. त्यामुळे सदर शैक्षणिक सामुग्री अधिक संपन्न झालेली असून,विद्यार्थी त्यांच्या व्यक्तिमत्त्वाचा सुसंवादी आणि सर्वांगीण विकास साधतील. परिणामतः विश्वस्तरीय मनुष्यबळाच्या गरजा पूर्ण करण्यात महाराष्ट्र राज्य अग्रेसर राहील व पर्यायाने राष्ट्रनिर्मिती करीता निश्चितच हातभार लागेल, असा मला विश्वास आहे.

अभियांत्रिकी पदविका अभ्यासक्रमातील विषयांची मराठी-इंग्रजी (द्विभाषिक) शैक्षणिक सामुग्री बनविण्यासाठी अध्यापक व सुकाणू समितीचे सदस्य यांनी दर्शविलेले समर्पण व वचनबद्धता कौतुकास पात्र आहे, या सर्वांचे मी मनःपूर्वक अभिनंदन करतो!

(डॉ. प्रमोद नाईक)
संचालक
म. रा. तंत्र शिक्षण मंडळ, मुंबई

अनुक्रमणिका

अ. नु.	युनिटचे नाव	पृष्ठ क्रमांक
1	सॉफ्टवेअर डेव्हलोपमेंट प्रोसेस (Software Development Process)	1
2	सॉफ्टवेअर रिक्वारमेन्ट इंजिनिअरिंग (Software Requirement Engineering)	18
3	सॉफ्टवेअर मॉडेलिंग आणि डिझाइन (Software Modeling and Design)	46
4	सॉफ्टवेअर प्रोजेक्ट कॉस्ट एस्टिमेशन (Software Project Cost Estimation)	83
5	सॉफ्टवेअर प्रोजेक्ट मॅनेजमेंट (Software Project Management)	95
6	सॉफ्टवेअर क्वालिटीअशुरन्स (Software Quality Assurance)	116

युनिट-1
सॉफ्टवेअर डेव्हलोपमेंट प्रोसेस
(Software Development Process)

विषय निष्पत्ती (Course Outcome):

CO1: योग्य सॉफ्टवेअर डेव्हलोपमेंट प्रोसेस मॉडेल निवडा.

घटक निष्पत्ती (Theory Learning Outcome - TLO):

1. दिलेल्या सॉफ्टवेअर ॲप्लिकेशनसाठी मानकांशी जुळणारे ऍट्रिब्यूट निवडा.
2. दिलेल्या समस्येसाठी संबंधित सॉफ्टवेअर सोल्यूशन सुचवा.
3. दिलेल्या समस्येसाठी संबंधित सॉफ्टवेअर प्रोसेस मॉडेल निवडा.
4. एजाईल (Agile) डेव्हलोपमेंट प्रोसेसमध्ये संबंधित ॲक्टिव्हिटी सुचवा.

1.1 सॉफ्टवेअर वैशिष्ट्ये (Software characteristics)

सॉफ्टवेअर : सॉफ्टवेअर म्हणजे संगणक चालवण्यासाठी आणि विशिष्ट कार्ये करण्यासाठी वापरल्या जाणाऱ्या सूचना, डेटा किंवा प्रोग्राम्सचा संग्रह. ते हार्डवेअरच्या विरुद्ध आहे, जे संगणकाचे भौतिक भाग आहे. सॉफ्टवेअर म्हणजे संगणक प्रोग्राम आणि संबंधित डेटाचा संग्रह जो संगणकाला कसे काम करायचे ते सांगतो.

ते वापरकर्ता आणि हार्डवेअरमधील पूल म्हणून काम करते - ऑपरेटिंग सिस्टम चालवण्यापासून ते संगीत वाजवणे किंवा दस्तऐवज संपादित करण्यापर्यंतची कामे सक्षम करते.

सॉफ्टवेअर म्हणजे: 1) सूचना (कॉम्प्युटर प्रोग्राम) ज्या अंमलात आणल्यावर इच्छित वैशिष्ट्ये, कार्य आणि कार्यप्रदर्शन प्रदान करतात; 2) डेटा स्ट्रक्चर्स जे प्रोग्राम्सना माहिती योग्यरित्या हाताळण्यास सक्षम करतात आणि 3) हार्ड कॉपी आणि व्हर्च्युअल स्वरूपात वर्णनात्मक माहिती जी प्रोग्राम्सच्या ऑपरेशन आणि वापराचे वर्णन करते.

सॉफ्टवेअरची वैशिष्ट्ये (Characteristics of Software):

मानवांनी बनवलेल्या इतर गोष्टींपेक्षा सॉफ्टवेअर वेगळे बनवणाऱ्या वैशिष्ट्यांचे परीक्षण करणे महत्त्वाचे आहे. सॉफ्टवेअर हा भौतिक प्रणाली घटक नसून तार्किक आहे. म्हणून, सॉफ्टवेअरमध्ये हार्डवेअरपेक्षा बरीच वेगळी वैशिष्ट्ये आहेत:

1. **सॉफ्टवेअर विकसित किंवा अभियंता आहे; हे शास्त्रीय अर्थाने तयार केले जात नाही (Software is developed or engineered; it is not manufactured in the classical sense)**
   सॉफ्टवेअर डेव्हलपमेंट आणि हार्डवेअर मॅन्युफॅक्चरिंगमध्ये काही समानता असली तरी, काही ॲक्टिव्हिटीस वेगळे आहेत. दोन्ही क्रियाकलापांमध्ये, चांगल्या डिझाइनद्वारे उच्च क्वालिटी प्राप्त केली जाते, परंतु हार्डवेअरसाठी उत्पादन टप्पा सॉफ्टवेअरपेक्षा क्वालिटी समस्या निर्माण करू शकतो.
2. **सॉफ्टवेअर डझन्ट वेअरऑउट (The software doesn't wearout)**
   हार्डवेअर खराब व्हायला लागतं. सॉफ्टवेअरवर पर्यावरणाचा परिणाम होत नाही ज्यामुळे हार्डवेअर खराब होतं. जेव्हा हार्डवेअरचा एखादा भाग खराब होतो, तेव्हा तो सुट्या भागांनी बदलला जातो. सॉफ्टवेअरसाठी असे कोणतेही सुटे भाग नसतात. प्रत्येक सॉफ्टवेअरमधील बिघाड हे डिझाइनमधील (design) किंवा डिझाइन मशीन-एक्झिक्यूटेबल कोडमध्ये (machine-executable code) रूपांतरित करण्याच्या प्रक्रियेतील त्रुटी दर्शवतो. त्यामुळे, बदलांच्या विनंत्या पूर्ण करणाऱ्या सॉफ्टवेअर देखभालीची (software maintenance) कामे हार्डवेअर देखभालीपेक्षा खूप जास्त क्लिष्ट असतात. मात्र, सॉफ्टवेअर खराब होत नाही.

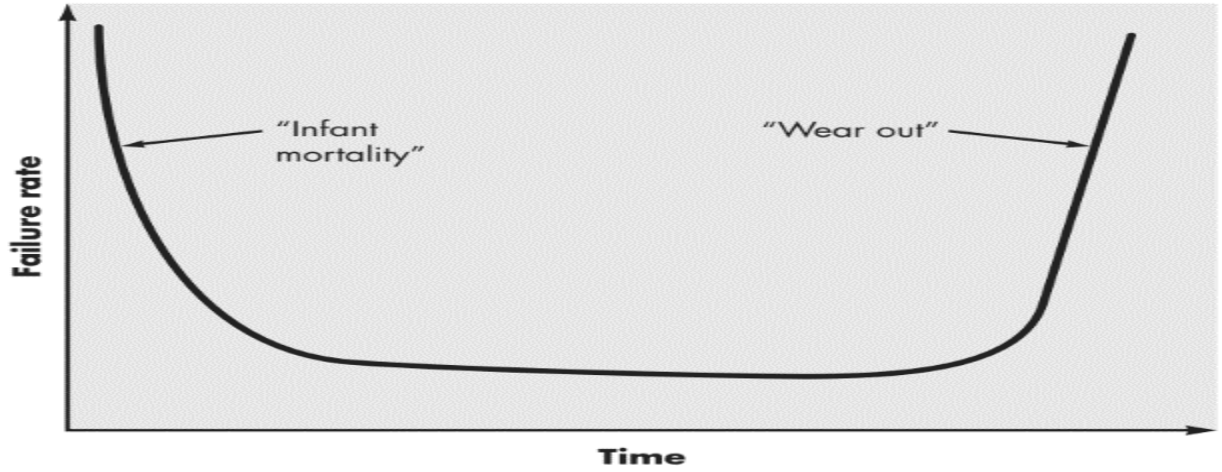


Fig. 1.1: फेल्युअर रेट फॉर हार्डवेअर (Failure rate for hardware)

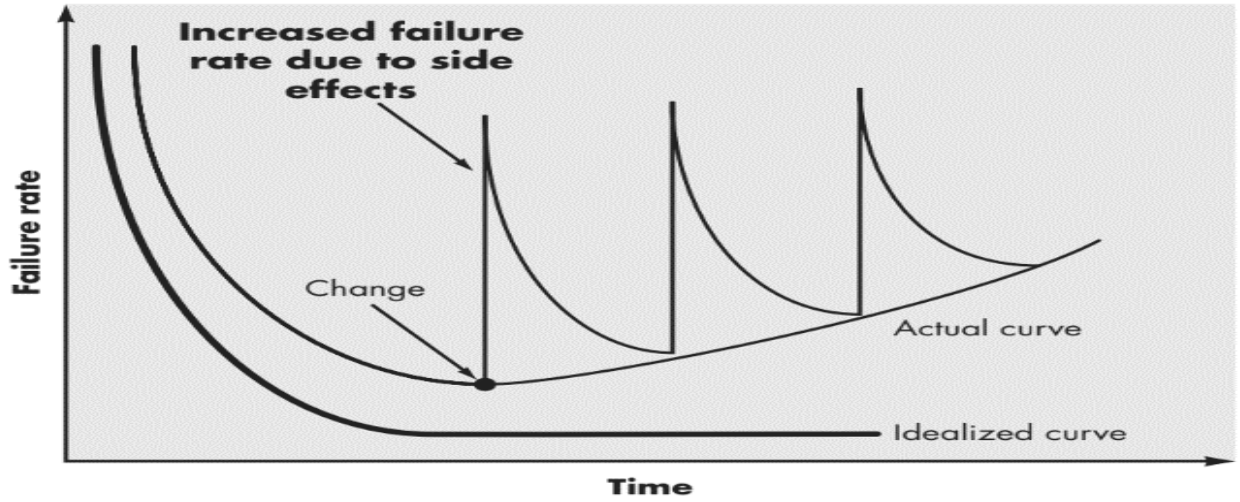


Fig. 1.2: फेल्युअर रेट फॉर सॉफ्टवेअर (Failure rate for software)

3. **जरी उद्योग घटक-आधारित कॅन्स्ट्रुक्शनकडे वाटचाल करत असला तरी, बहुतेक सॉफ्टवेअर कस्टम बिल्ट केले जात आहेत (Although the industry is moving toward component-based construction, most software continues to be custom built).**

सॉफ्टवेअर भागाचे प्लॅनिंग आणि अंमलबजावणी अशा प्रकारे केली पाहिजे की तो पुन्हा वापरला जाईल. विविध प्रकल्पांमध्ये. सध्याचे पुनर्वापरयोग्य विभाग प्रोग्रामरला पुनर्वापरयोग्य भागांपासून नवीन अनुप्रयोग बनवण्याची परवानगी देतात. हार्डवेअरमध्ये, घटकांचा पुनर्वापर हा इंजिनिअरिंग प्रक्रियेचा एक नैसर्गिक भाग आहे.

1.1.1 सॉफ्टवेअरचे प्रकार (Types of software)

1. **सिस्टम सॉफ्टवेअर (System Software):** इतर प्रोग्राम्सना सेवा देण्यासाठी लिहिलेल्या प्रोग्राम्सचा संग्रह. काही सिस्टम सॉफ्टवेअर (उदा. कंपायलर, एडिटर आणि फाइल मॅनेजमेंट युटिलिटीज) जटिल, परंतु निश्चित माहिती संरचनांवर प्रोसेस करतात. इतर सिस्टम ॲप्लिकेशन्स (उदा. ऑपरेटिंग सिस्टम घटक, ड्रायव्हर्स, नेटवर्किंग सॉफ्टवेअर, टेलिकम्युनिकेशन प्रोसेसर) मोठ्या प्रमाणात अनिश्चित डेटावर प्रोसेस करतात.
2. **ॲप्लिकेशन सॉफ्टवेअर (Application Software):** विशिष्ट बीसनेस गरजा पूर्ण करणारे स्वतंत्र कार्यक्रम. या क्षेत्रातील ॲप्लिकेशन्स बीसनेस किंवा तांत्रिक डेटा अशा प्रकारे प्रोसेस करतात ज्यामुळे बीसनेस ऑपरेशन्स किंवा व्यवस्थापन/तांत्रिक निर्णय घेणे सोपे होते. पारंपारिक डेटा प्रोसेसिंग ॲप्लिकेशन्स व्यतिरिक्त, ॲप्लिकेशन सॉफ्टवेअरचा वापर रिअल टाईममध्ये बीसनेस कार्ये नियंत्रित करण्यासाठी केला जातो (उदा., पॉइंट-ऑफ-सेल ट्रान्झॅक्शन प्रोसेसिंग, रिअल-टाईम मॅन्युफॅक्चरिंग प्रोसेस कंट्रोल).
3. **वेब अनुप्रयोग सॉफ्टवेअर (Web Applications Software):** ज्याला "वेबॲप्स" म्हणतात, ही नेटवर्क-केंद्रित सॉफ्टवेअर श्रेणी विविध प्रकारच्या ॲप्लिकेशन्समध्ये पसरलेली आहे. त्यांच्या सर्वात सोप्या स्वरूपात,

वेबॲप्स हे लिंक्ड हायपरटेक्स्ट फाइल्सच्या संचापेक्षा थोडे अधिक असू शकतात जे मजकूर आणि मर्यादित ग्राफिक्स वापरून माहिती सादर करतात. तथापि, वेब २.० उदयास येत असताना, वेबॲप्स अशा अत्याधुनिक संगणकीय वातावरणात विकसित होत आहेत जे अंतिम वापरकर्त्याला केवळ स्वतंत्र वैशिष्ट्ये, संगणकीय कार्ये आणि सामग्री प्रदान करत नाहीत तर कॉर्पोरेट डेटाबेस आणि बीसनेस अनुप्रयोगांसह देखील एकत्रित केले जातात.

4. **एम्बेडेड केलेले सॉफ्टवेअर (Embedded Software):** उत्पादन किंवा सिस्टममध्ये राहते आणि अंतिम वापरकर्त्यासाठी आणि सिस्टमसाठी वैशिष्ट्ये आणि कार्ये अंमलात आणण्यासाठी आणि नियंत्रित करण्यासाठी वापरले जाते. एम्बेडेड सॉफ्टवेअर मर्यादित आणि गूढ कार्ये करू शकते (उदा. मायक्रोवेव्ह ओव्हनसाठी की पॅड नियंत्रण) किंवा महत्त्वपूर्ण कार्य आणि नियंत्रण क्षमता प्रदान करू शकते (उदा., इंधन नियंत्रण, डॅशबोर्ड डिस्प्ले आणि ब्रेकिंग सिस्टम यासारख्या ऑटोमोबाईलमधील डिजिटल कार्ये).
5. **रिझर्व्हशन सॉफ्टवेअर (Reservation Software):** रिझर्व्हशन प्रणाली प्रामुख्याने माहिती संग्रहित आणि पुनर्प्राप्त करण्यासाठी आणि हवाई प्रवास, कार भाड्याने, हॉटेल किंवा इतर क्रियाकलापांशी संबंधित व्यवहार करण्यासाठी वापरली जाते.
6. **बीझनेस सॉफ्टवेअर (Business Software):** सॉफ्टवेअरची ही श्रेणी बीझनेस अनुप्रयोगांना समर्थन देण्यासाठी वापरली जाते आणि सॉफ्टवेअरची सर्वाधिक प्रमाणात वापरली जाणारी श्रेणी आहे. इन्व्हेंटरी मॅनेजमेंट, अकाउंट्स, बँकिंग, रुग्णालये, शाळा, स्टॉक मार्केट इत्यादींसाठी सॉफ्टवेअर उदाहरणे आहेत.
7. **करमणूक सॉफ्टवेअर (Entertainment Software):** शिक्षण आणि करमणूक सॉफ्टवेअर शैक्षणिक एजन्सींसाठी एक शक्तिशाली साधन प्रदान करते. विशेषत: जे लहान मुलांना शिक्षित करण्याच्या बाबतीत व्यवहार करतात. संगणक गेम्स, शैक्षणिक खेळ इ. सारखे सॉफ्टवेअर.
8. **आर्टीफिशीअल इंटेलिजन्स सॉफ्टवेअर (Artificial Intelligence Software):** गणना किंवा सरळ पुढे विश्लेषणासाठी सक्षम नसलेल्या जटिल समस्या सोडवण्यासाठी नॉन-न्यूमेरिकल अल्गोरिदमचा वापर करते. या क्षेत्रातील अनुप्रयोगांमध्ये रोबोटिक्स, तज्ञ सिस्टिम , पॅटर्न ओळख (प्रतिमा आणि आवाज), कृत्रिम तंत्रिका नेटवर्क, प्रमेय सिद्ध करणे आणि गेम प्ले करणे यांचा समावेश आहे.
9. **वैज्ञानिक सॉफ्टवेअर (Scientific Software):** वैज्ञानिक आणि अभियांत्रिकी सॉफ्टवेअर विशिष्ट कार्ये करण्यासाठी वैज्ञानिक किंवा अभियांत्रिकी वापरकर्त्याच्या गरजा भागवते. मॅटलाब, ऑटोकॅड, ऑक्टेव्ह इ. सारखी उदाहरणे
10. **उत्पादन-लाइन सॉफ्टवेअर (Product Line Software):** अनेक वेगवेगळ्या ग्राहकांद्वारे वापरण्यासाठी विशिष्ट क्षमता प्रदान करण्यासाठी डिझाइन केलेले. उत्पादन-लाइन सॉफ्टवेअर मर्यादित आणि गूढ बाजारपेठेवर लक्ष केंद्रित करू शकते (उदा., इन्व्हेंटरी नियंत्रण उत्पादने) किंवा मोठ्या प्रमाणात ग्राहक बाजारपेठांना संबोधित करू शकते (उदा., वर्ड प्रोसेसिंग, स्प्रेडशीट्स, संगणक ग्राफिक्स, मल्टीमीडिया, मनोरंजन, डेटाबेस व्यवस्थापन आणि वैयक्तिक आणि व्यावसायिक आर्थिक अनुप्रयोग).

1.2 प्रोसेस सॉफ्टवेअर अभियांत्रिकी: एक स्तरित दृष्टिकोन - प्रोसेस, पद्धती आणि साधने

(The Process Software Engineering: A Layered Approach-Process, Methods and Tools)

सॉफ्टवेअर इंजिनिअरिंग ही एक स्तरित (लेयर्ड) तंत्रज्ञान आहे . Fig. 1.3 चा संदर्भ ,घेतल्यास कोणताही इंजिनिअरिंग पद्धत (सॉफ्टवेअर अभियांत्रिकीसह) ही गुणवत्ता सुनिश्चित करण्याच्या संस्थात्मक बांधिलकीवर आधारित असते.
टोटल क्वालिटी मॅनेजमेंट (TQM), सिक्स सिग्मा आणि अशा इतर तत्त्वज्ञानांमुळे सातत्यपूर्ण प्रोसेसवाढीस चालना मिळते, आणि हीच प्रोसेस संस्कृती अखेरीस सॉफ्टवेअर अभियांत्रिकीसाठी अधिक प्रभावी पद्धतींच्या विकासास कारणीभूत ठरते.

लेयर्स खालीलप्रमाणे आहेत:

Fig. 1.3: सॉफ्टवेअर इंजिनीअरिंग ऐज अ लेयर्ड ॲप्रोच (Software Engineering as a Layered Approach)

1. **क्वालिटी फोकस लेअर (Quality Focus):**
   - सॉफ्टवेअर इंजिनिअरिंग ला समर्थन देणारा पाया म्हणजे क्वालिटी फोकस.
   - सॉफ्टवेअर इंजिनिअरिंग चा पाया म्हणजे प्रोसेस लेअर.
   - सॉफ्टवेअर इंजिनिअरिंग प्रोसेस ही गोंद आहे जी तंत्रज्ञानाच्या स्तरांना एकत्र ठेवते आणि संगणक सॉफ्टवेअरचा तर्कसंगत आणि वेळेवर विकास सक्षम करते.
2. **प्रोसेस लेअर (Process)**:
   - प्रोसेस एक फ्रेमवर्क परिभाषित करते जी सॉफ्टवेअर इंजिनिअरिंग तंत्रज्ञानाच्या प्रभावी वितरणासाठी स्थापित केली पाहिजे.
   - सॉफ्टवेअर प्रोसेस सॉफ्टवेअर प्रकल्पांच्या व्यवस्थापन नियंत्रणासाठी आधार बनवते आणि तांत्रिक मेथड कोणत्या संदर्भात लागू केल्या जातात, कार्य उत्पादने (मॉडेल, दस्तऐवज, डेटा, अहवाल, फॉर्म इ.) तयार केली जातात, टप्पे स्थापित केले जातात, क्वालिटी सुनिश्चित केली जाते आणि बदल योग्यरित्या व्यवस्थापित केला जातो.
3. **मेथड लेअर (Method)**:
   - सॉफ्टवेअर इंजिनिअरिंग मेथड सॉफ्टवेअर तयार करण्यासाठी तांत्रिक मेथड प्रदान करतात. मेथड मध्ये कम्युनिकेशन, रीक्वॅयरमेन्ट अनालीसीस, डेटा मॉडेलिंग, प्रोग्रॅम कन्स्ट्रुक्शन, टेस्टिंग व सपोर्ट यासारख्या विस्तृत कार्यांचा समावेश आहे.
4. **टूल्स लेअर (Tools):**
   - सॉफ्टवेअर अभियांत्रिकी पद्धती तंत्रज्ञानाच्या प्रत्येक क्षेत्रावर नियंत्रण ठेवणाऱ्या मूलभूत तत्त्वांच्या संचावर अवलंबून असतात आणि मॉडेलिंग क्रियाकलाप आणि इतर वर्णनात्मक तंत्रांचा समावेश करतात.
   - सॉफ्टवेअर अभियांत्रिकी साधने प्रोसेस आणि पद्धतींसाठी स्वयंचलित किंवा अर्ध स्वयंचलित समर्थन प्रदान करतात.
   - जेव्हा टूल्स एकत्रित केली जातात जेणेकरून एका टूल्स द्वारे तयार केलेली माहिती दुसऱ्या टूल्स द्वारे वापरली जाऊ शकते, तेव्हा सॉफ्टवेअर विकासाच्या समर्थनासाठी एक सिस्टिम, ज्याला संगणक-सहाय्यित सॉफ्टवेअर इंजिनिअरिंग म्हणतात, स्थापित केली जाते.

1.3 सॉफ्टवेअर डेव्हलपमेंट फ्रेमवर्क (Software Development Framework):

सॉफ्टवेअर इंजिनिअरिंग साठी एक जेनेरिक प्रॉसेस फ्रेमवर्क पाच फ्रेमवर्क ॲक्टिव्हिटीसची व्याख्या करते - कम्युनिकेशन, प्लॅनिंग , मॉडेलिंग, कन्स्ट्रुकशन आणि डिप्लॉयमेंट . याव्यतिरिक्त, संपूर्ण प्रक्रियेत अम्ब्रेला ॲक्टिव्हिटीसची संच - प्रोजक्ट ट्रॅकिंग आणि नियंत्रण, रिस्क व्यवस्थापन, क्वालिटी हमी, कॉन्फिगरेशन व्यवस्थापन, तांत्रिक पुनरावलोकने आणि इतर - लागू केले जातात. हे मुख्य ॲक्टिव्हिटीस आहेत जे जवळजवळ प्रत्येक सॉफ्टवेअर विकास प्रक्रियेत होतात, मॉडेल काहीही असो (उदा., वॉटरफॉल, ॲजाइल, स्पायरल).

1. **कम्युनिकेशन (Communication):**
   - ग्राहकाला काय हवे आहे ते समजून घ्या•
   - भागधारकांच्या ,रीक्वॅयरमेन्ट एकत्र करणेमुलाखती आणि बैठका समाविष्ट आहेत.

2. प्लॅनिंग (Planning):

- संसाधने, वेळ आणि खर्चाचा अंदाज लावा.
- प्रोजक्ट वेळापत्रक, टप्पे आणि रिस्क परिभाषित करा.

3. मॉडेलिंग (Modeling):

सॉफ्टवेअरची रचना आणि वर्तन चांगल्या प्रकारे समजून घेण्यासाठी डिझाइन आणि मॉडेल्स तयार करा.
यात समाविष्ट आहे:

- रीक्वॅयरमेन्ट विश्लेषण
- आर्किटेक्चरल डिझाइन
- डेटा आणि प्रोसेस मॉडेलिंग

4. कॅन्स्ट्रुक्शन (Construction):

- सॉफ्टवेअर कोड करा आणि युनिट आणि इंटिग्रेशन टेस्ट करा.
- डिझाइनला एक्झिक्युटेबल प्रोग्राममध्ये रूपांतरित करते.

5. डिप्लॉयमेंट (Deployment):

- वापरकर्त्यांना सॉफ्टवेअर वितरित करा.
- स्थापना, समर्थन, प्रशिक्षण आणि देखभाल यांचा समावेश असू शकतो.

अम्ब्रेला ॲक्टिव्हिटीस (Umbrella Activities):

सॉफ्टवेअर इंजिनिअरिंग प्रोसेस फ्रेमवर्क ॲक्टिव्हिटीस अनेक **अम्ब्रेला** क्रियाकलापांनी पूरक असतात. सर्वसाधारणपणे, **अम्ब्रेला** ॲक्टिव्हिटीस संपूर्ण सॉफ्टवेअर प्रकल्पात लागू केले जातात आणि सॉफ्टवेअर टीमला प्रगती, क्वालिटी , बदल आणि रिस्क व्यवस्थापित आणि नियंत्रित करण्यास मदत करतात. सामान्य **अम्ब्रेला** क्रियाकलापांमध्ये हे समाविष्ट आहे:

1. **सॉफ्टवेअर प्रोजक्ट ट्रॅकिंग आणि नियंत्रण (Software project tracking and control)** - सॉफ्टवेअर टीमला प्रोजक्ट योजनेच्या प्रगतीचे मूल्यांकन करण्यास आणि वेळापत्रक राखण्यासाठी आवश्यक असलेली कोणतीही कारवाई करण्यास अनुमती देते.
2. **रिस्क व्यवस्थापन (Risk management)** - प्रकल्पाच्या परिणामावर किंवा उत्पादनाच्या गुणवत्तेवर परिणाम करू शकणाऱ्या जोखमींचे मूल्यांकन करते.
3. **सॉफ्टवेअर क्वालिटी हमी (Software quality assurance) -** सॉफ्टवेअर क्वालिटी सुनिश्चित करण्यासाठी आवश्यक असलेल्या क्रियाकलापांची व्याख्या आणि अंमलबजावणी करते.
4. **तांत्रिक पुनरावलोकने (Technical reviews) -** सॉफ्टवेअर इंजिनिअरिंग कामाच्या उत्पादनांचे मूल्यांकन करून पुढील क्रियाकलापांमध्ये त्रुटी शोधून काढणे आणि त्या दूर करणे.
5. **मोजमाप (Measurement) -** प्रोसेस, प्रोजक्ट आणि उत्पादन उपाय परिभाषित करते आणि गोळा करते जे भागधारकांच्या गरजा पूर्ण करणारे सॉफ्टवेअर वितरित करण्यात टीमला मदत करते; इतर सर्व फ्रेमवर्क आणि छत्री क्रियाकलापांसह एकत्रितपणे वापरले जाऊ शकते.
6. **सॉफ्टवेअर कॉन्फिगरेशन व्यवस्थापन (Software configuration management)-** संपूर्ण सॉफ्टवेअर प्रक्रियेतील बदलांचे परिणाम व्यवस्थापित करते.
7. **पुनर्वापरयोग्यता व्यवस्थापन (Reusability management) -** कामाच्या उत्पादनाच्या पुनर्वापरासाठी (सॉफ्टवेअर घटकांसह) निकष परिभाषित करते आणि पुन्हा वापरता येण्याजोगे घटक साध्य करण्यासाठी यंत्रणा स्थापित करते.
8. **कामाच्या उत्पादनाची तयारी आणि उत्पादन (Work product preparation and production) -** मॉडेल, दस्तऐवज, लॉग, फॉर्म आणि सूची यासारख्या कामाच्या उत्पादनांची निर्मिती करण्यासाठी आवश्यक असलेल्या ॲक्टिव्हिटीसची समावेश करते.

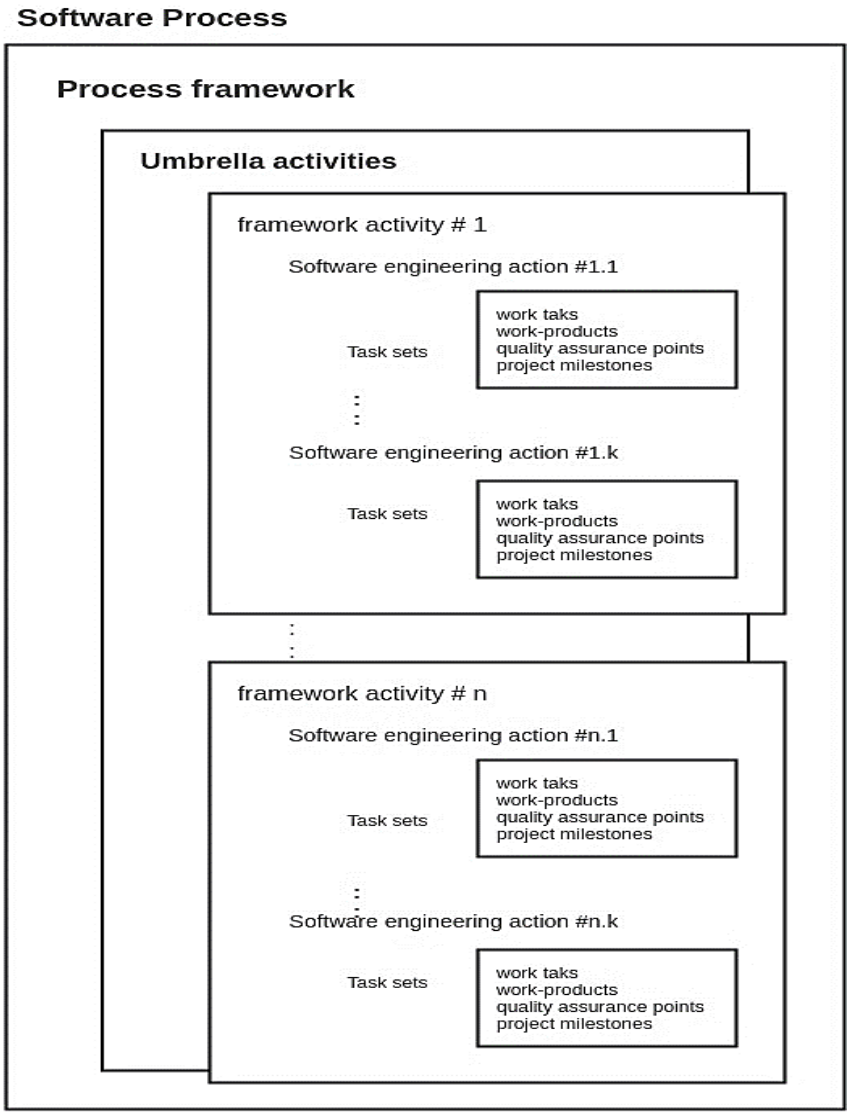


Fig. 1.4: सॉफ्टवेअर डेव्हलपमेंट फ्रेमवर्क (Software development framework)

1.4 सॉफ्टवेअर प्रोसेस मॉडेल: वॉटरफॉल मॉडेल (Software Process Model: Waterfall Model)

असे काही वेळा येतात जेव्हा समस्येच्या रीक्वॅयरमेन्ट चांगल्या प्रकारे समजल्या जातात—जेव्हा काम संवादातून वाजवी रेषीय मेथड ने डिप्लॉयमेंट द्वारे वाहते. ही परिस्थिती कधीकधी तेव्हा येते जेव्हा विद्यमान सिस्टिम मध्ये सु-परिभाषित रूपांतरे किंवा सुधारणा कराव्या लागतात (उदा., सरकारी नियमांमधील बदलांमुळे अनिवार्य केलेल्या अकाउंटिंग सॉफ्टवेअरशी जुळवून घेणे). हे मर्यादित संख्येने नवीन विकास प्रयत्नांमध्ये देखील येऊ शकते, परंतु जेव्हा रीक्वॅयरमेन्ट चांगल्या प्रकारे परिभाषित केल्या जातात आणि वाजवी स्थिर असतात. वॉटरफॉल मॉडेल, ज्याला कधीकधी क्लासिक लाइफ सायकल म्हणतात, सॉफ्टवेअर डेव्हलपमेंटसाठी एक पद्धतशीर, अनुक्रमिक दृष्टिकोन सुचवते जो ग्राहकांच्या आवश्यकतांच्या विशिष्टतेपासून सुरू होतो आणि प्लॅनिंग , मॉडेलिंग, कॉन्स्ट्रक्शन आणि डिप्लॉयमेंट द्वारे प्रगती करतो, पूर्ण झालेल्या सॉफ्टवेअरच्या सतत समर्थनात परिणत होतो.

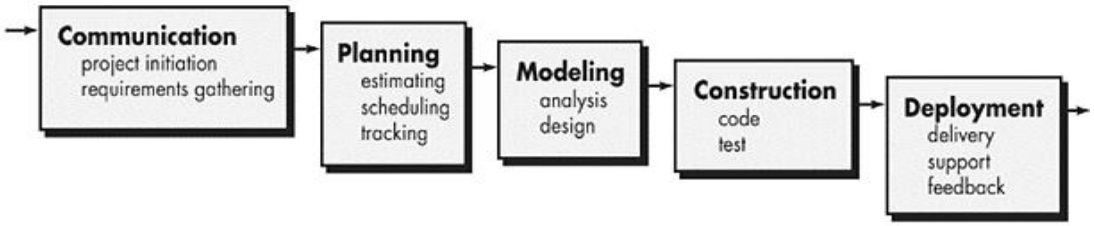


Fig. 1.5: वॉटरफॉल मॉडेल (Waterfall Model)

वॉटरफॉल मॉडेलचे टप्पे:

1. **कम्युनिकेशन :**
   उपक्रम: प्रकल्पाची सुरुवात, रीक्वॅयरमेन्ट गोळा करणे.
   उद्देश: हा सुरुवातीचा टप्पा आहे जिथे भागधारक, वापरकर्ते आणि विकासक सॉफ्टवेअर रीक्वॅयरमेन्ट समजून घेण्यासाठी कम्युनिकेशन साधतात. सिस्टमने काय करावे हे परिभाषित करण्यासाठी स्पष्ट दस्तऐवजीकरण तयार केले जाते.
2. **प्लॅनिंग :**
   उपक्रम: अंदाज, वेळापत्रक, ट्रॅकिंग
   उद्देश: या टप्प्यात, प्रोजक्ट टीम एक तपशीलवार प्रोजक्ट प्लॅन विकसित करते. यामध्ये वेळ आणि खर्चाचा अंदाज, वेळापत्रक तयार करणे, संसाधनांचे वाटप करणे आणि प्रोजक्ट ट्रॅकवर राहतो याची खात्री करण्यासाठी टप्पे निश्चित करणे समाविष्ट आहे.
3. **मॉडेलिंग :**
   उपक्रम: विश्लेषण, डिझाइन
   उद्देश: हा टप्पा सिस्टम आर्किटेक्चर आणि डिझाइनवर लक्ष केंद्रित करतो. विश्लेषक डेटा आणि प्रोसेस मॉडेल तयार करतात (जसे की DFDs, ER आकृत्या), आणि डिझाइनर सिस्टम कसे कार्य करेल आणि कसे दिसेल हे परिभाषित करतात.
4. **कन्स्ट्रक्शन:**
   उपक्रम: कोड, टेस्टिंग
   उद्देश: येथे प्रत्यक्ष विकास होतो. विकासक पूर्वीच्या डिझाइन दस्तऐवजांवर आधारित कोड लिहितात. कोडिंग केल्यानंतर, बग ओळखण्यासाठी आणि दुरुस्त करण्यासाठी सिस्टमची कसून टेस्ट केली जाते.
5. **डिप्लॉयमेंट :**
   उपक्रम: डिलिव्हरी , समर्थन, फीडबॅक
   उद्देश: अंतिम उत्पादन क्लायंटला वितरित केले जाते. देखभालीसाठी समर्थन प्रदान केले जाते आणि भविष्यातील पुनरावृत्ती सुधारण्यासाठी किंवा बदल हाताळण्यासाठी फीडबॅक गोळा केला जातो.

फायदे (Advantages) :

1. सोपे आणि समजण्यास सोपे आणि वापरण्यास सोपे.
2. स्पष्ट, निश्चित रीक्वॅयरमेन्ट असलेल्या लहान प्रकल्पांसाठी चांगले कार्य करते.
3. त्याच्या कडकपणामुळे व्यवस्थापित करणे सोपे (प्रत्येक टप्प्यात विशिष्ट डिलिव्हरेबल्स असतात).

तोटे (Disadvantages) :

1. बदलांना लवचिक नसणे: मागील टप्प्यावर परत जाणे कठीण.
2. दीर्घकालीन किंवा गुंतागुंतीच्या प्रकल्पांसाठी खराब मॉडेल.
3. समस्यांचा उशिरा शोध (अंमलबजावणीनंतर टेस्ट होते).
4. रीक्वॅयरमेन्ट पूर्ण झाल्यानंतर मर्यादित ग्राहक सहभाग.

1.5 वाढीव प्रोसेस मॉडेल: आर एडी मॉडेल (Incremental Process Model: RAD Model)

आरएडी मॉडेल (Rapid Application Development model):

रॅपिड ॲप्लिकेशन डेव्हलपमेंट (आरएडी) मॉडेल ही एक सॉफ्टवेअर डेव्हलपमेंट पद्धत आहे जी संपूर्ण विकास चक्रात वापरकर्त्यांच्या सक्रिय सहभागासह अनुप्रयोगांच्या क्वीक आणि पुनरावृत्ती वितरणाला प्राधान्य देते. वॉटरफॉल सारख्या पारंपारिक सॉफ्टवेअर मॉडेल्सच्या मर्यादांना प्रतिसाद म्हणून 1980 मध्ये ते सादर करण्यात आले, जे कठोर आणि वेळखाऊ आहेत. आरएडी मॉडेल विकास प्रक्रियेला पाच प्रमुख टप्प्यांमध्ये विभागते: बीसनेस मॉडेलिंग, डेटा मॉडेलिंग, प्रोसेस मॉडेलिंग, ॲप्लिकेशन जनरेशन आणि टेस्टिंग अँड टर्नओव्हर.

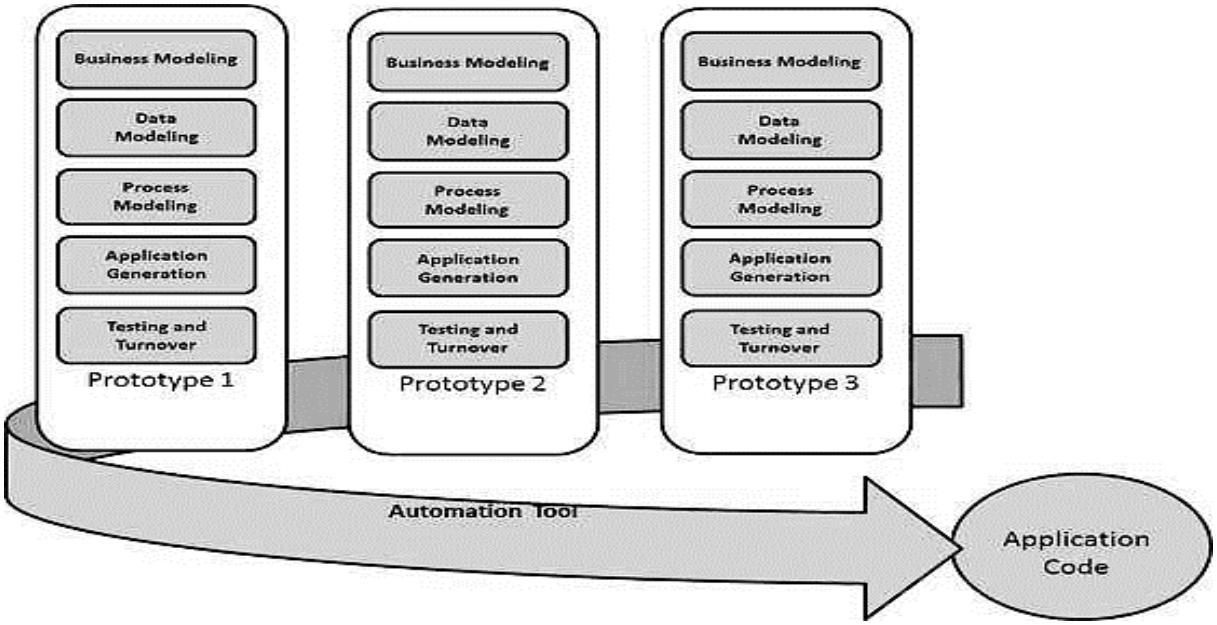


Fig. 1.6: आर ए डी मॉडेल (RAD Model)

RADमॉडेलचे टप्पे:

1. **बीसनेस मॉडेलिंग**
   - बीसनेस कार्ये आणि माहितीचा प्रवाह समजून घ्या.
2. **डेटा मॉडेलिंग**
   - डेटा ऑब्जेक्ट्स आणि संबंध ओळखा आणि डिझाइन करा.
3. **प्रोसेस मॉडेलिंग**
   - डेटा ऑब्जेक्ट्स आणि बीसनेस प्रोसेस हाताळण्यासाठी तर्कशास्त्र परिभाषित करा.
4. **अँप्लिकेशन जनरेशन**
   - मॉडेल्सना वास्तविक अनुप्रयोग कोडमध्ये द्रुतपणे रूपांतरित करण्यासाठी स्वयंचलित साधनांचा वापर करा.
5. **टेस्ट आणि टूर्नओवर**
   - वैयक्तिक घटकांची टेस्ट घ्या आणि त्यांना कार्यरत सिस्टिम मध्ये एकत्रित करा.
   - वापरकर्ता फीडबॅक सतत गोळा केला जातो आणि बदल क्वीक लागू केले जातात.

फायदे (Advantages) :

1. कार्यात्मक सॉफ्टवेअरची क्वीक डिलिव्हरी .
2. ग्राहक फीडबॅक आणि सहकार्याला प्रोत्साहन देते.
3. विकास वेळ आणि खर्च कमी करते.
4. तुकड्यांमध्ये बांधता येणाऱ्या मॉड्यूलर सिस्टमसाठी आदर्श.

तोटे (Disadvantages) :

1. मोठ्या प्रमाणात किंवा जटिल प्रकल्पांसाठी योग्य नाही.
2. कुशल विकासक आणि डिझाइनर्सची रीक्वॅयरमेन्ट आहे.
3. क्वीक प्रोटोटाइपिंग साधनांवर जास्त अवलंबून राहणे.
4. खराब दस्तऐवजीकरणामुळे देखभाल समस्या उद्भवू शकतात.

1.6 उत्क्रांती प्रोसेस मॉडेल: प्रोटोटाइपिंग मॉडेल, स्पायरल मॉडेल (Evolutionary Process Models: Prototyping model, Spiral model)

1.6.1 प्रोटोटाइपिंग मॉडेल (Prototyping model):

बहुतेकदा, ग्राहक सॉफ्टवेअरसाठी सामान्य उद्दिष्टांचा संच परिभाषित करतो, परंतु फंक्शन्स आणि वैशिष्ट्यांसाठी तपशीलवार रीक्वॅयरमेन्ट ओळखत नाही. इतर प्रकरणांमध्ये, विकासकाला अल्गोरिथमची कार्यक्षमता, ऑपरेटिंग सिस्टमची अनुकूलता किंवा मानव-मशीन परस्परसंवादाचे स्वरूप याबद्दल खात्री नसते. या आणि इतर अनेक परिस्थितींमध्ये, प्रोटोटाइपिंग पॅराडाइम सर्वोत्तम दृष्टिकोन देऊ शकते. क्वीक डिझाइन प्रोटोटाइपच्या बांधकामाकडे नेतो. प्रोटोटाइपचा वापर भागधारकांद्वारे केला जातो आणि त्याचे मूल्यांकन केले जाते, जे फीडबॅक प्रदान करतात जे रीक्वॅयरमेन्ट अधिक परिष्कृत करण्यासाठी वापरले जातात. विविध भागधारकांच्या गरजा पूर्ण करण्यासाठी प्रोटोटाइप ट्यून केल्यावर पुनरावृत्ती होते, त्याच वेळी तुम्हाला काय करावे लागेल हे चांगल्या प्रकारे समजून घेण्यास सक्षम करते. आदर्शपणे, प्रोटोटाइप सॉफ्टवेअर रीक्वॅयरमेन्ट ओळखण्यासाठी एक यंत्रणा म्हणून काम करते. जर कार्यरत प्रोटोटाइप तयार करायचा असेल, तर तुम्ही विद्यमान प्रोग्राम तुकड्यांचा वापर करू शकता किंवा सक्षम करणारी टूल्स (उदा., रिपोर्ट जनरेटर आणि विंडो व्यवस्थापक) लागू करू शकता. प्रोटोटाइप "पहिली सिस्टिम " म्हणून काम करू शकते. जरी काही प्रोटोटाइप "थ्रोवेज" म्हणून बनवले गेले असले तरी, इतर उत्क्रांतीवादी आहेत या अर्थाने की प्रोटोटाइप हळूहळू प्रत्यक्ष सिस्टिम मध्ये विकसित होतो.

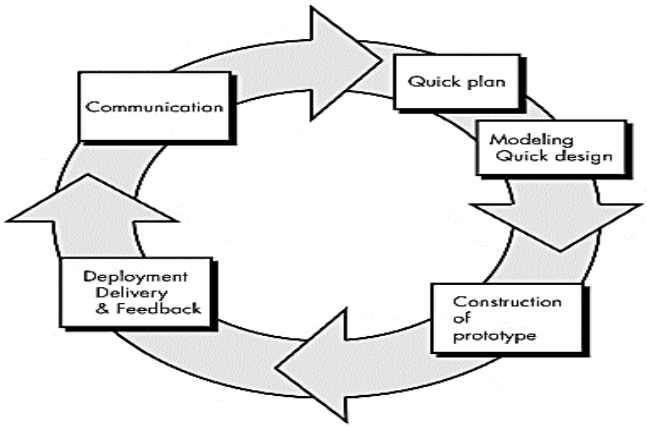


Fig. 1.7: प्रोटोटाइप मॉडेल (Prototype Model)

प्रोटोटाइपिंग मॉडेलचे टप्पे (Phases of the Prototyping Model) :

1. **कम्युनिकेशन (Communication):**
   उद्देश: व्यवसायाच्या गरजा समजून घ्या आणि भागधारकांकडून प्रारंभिक रीक्वॅयरमेन्ट गोळा करा.
   ॲक्टिव्हिटीस : बैठका, विचारमंथन, वापरकर्त्यांच्या मुलाखती.
   परिणाम: वापरकर्त्याला सॉफ्टवेअरमध्ये काय हवे आहे याची स्पष्ट समज.
2. **क्वीक प्लॅन (Quick Plan):**
   उद्देश: अल्पकालीन विकास योजनेची रूपरेषा तयार करा.
   ॲक्टिव्हिटीस : वैशिष्ट्यांना प्राधान्य द्या, कार्ये शेड्यूल करा, भूमिका नियुक्त करा.
   परिणाम: क्वीक सुरुवात करण्यासाठी एक हलकी आणि लवचिक प्रोजक्ट प्लॅन.
3. **मॉडेलिंग / क्वीक डिझाइन (Modelling/Quick Design) :**
   उद्देश: सिस्टमची क्वीक, संकल्पनात्मक रचना किंवा मॉडेल तयार करा.
   ॲक्टिव्हिटीस : मॉक-अप, फ्लो डायग्राम किंवा क्वीक UI डिझाइन तयार करा.
   परिणाम: प्रोटोटाइप विकासाचे मार्गदर्शन करणारे दृश्यमान किंवा कार्यात्मक डिझाइन.
4. **प्रोटोटाइपचे कन्स्ट्रक्शन (Construction of prototype):**
   उद्देश: डिझाइनवर आधारित कार्यरत प्रोटोटाइप विकसित करा.
   ॲक्टिव्हिटीस : वैशिष्ट्यांचे क्वीक कोडिंग आणि युनिट टेस्टिंग .

परिणाम: सिस्टमची कार्यात्मक आवृत्ती किंवा वापरकर्त्याच्या टेस्टिंग साठी एक नमुना.

5. **डिप्लॉयमेंट , डिलिव्हरी आणि फीडबॅक (Deployment, Delivery and Feedback):**
   उद्देश: वापरकर्त्याला नमुना वितरित करणे आणि फीडबॅक गोळा करणे.
   उपक्रम: प्रात्यक्षिके, फीडबॅक संकलन, परिष्करण.
   परिणाम: पुढील पुनरावृत्तीमध्ये नमुना सुधारण्यासाठी अंतर्दृष्टी.

फायदे (Advantages) :

1. अस्पष्ट किंवा बदलत्या रीक्वॅयरमेन्ट स्पष्ट करण्यास मदत करते.
2. वापरकर्त्याचा सहभाग आणि समाधान वाढवते.
3. डिझाइन आणि वापरण्यायोग्यतेच्या समस्यांचे लवकर निदान.
4. चुकीची सिस्टिम तयार करण्याचा धोका कमी करते.

तोटे (Disadvantages):

1. व्याप्ती कमी होऊ शकते (वापरकर्ते बदलांची विनंती करत राहतात).
2. अपूर्ण प्रोटोटाइपमुळे खोट्या अपेक्षा निर्माण होऊ शकतात.
3. कामगिरी-महत्वाच्या प्रणालींसाठी योग्य नाही.
4. वारंवार ग्राहकांशी कम्युनिकेशन आणि विकासक लवचिकता आवश्यक आहे.

1.6.2 स्पायरल मॉडेल (Spiral model):

स्पायरल मॉडेल हे एक उत्क्रांतीवादी सॉफ्टवेअर प्रोसेस मॉडेल आहे जे प्रोटोटाइपिंगच्या पुनरावृत्ती स्वरूपाला वॉटरफॉल मॉडेलच्या नियंत्रित आणि पद्धतशीर पैलूंसह जोडते. ते सॉफ्टवेअरच्या वाढत्या प्रमाणात अधिक पूर्ण आवृत्त्यांच्या क्रीक विकासाची क्षमता प्रदान करते. स्पायरल मॉडेल वापरून, सॉफ्टवेअर उत्क्रांतीवादी प्रकाशनांच्या मालिकेत विकसित केले जाते. सुरुवातीच्या पुनरावृत्ती दरम्यान, प्रकाशन एक मॉडेल किंवा प्रोटोटाइप असू शकते. नंतरच्या पुनरावृत्ती दरम्यान, इंजिनिअर केलेल्या सिस्टिम च्या वाढत्या प्रमाणात अधिक पूर्ण आवृत्त्या तयार केल्या जातात. स्पायरल मॉडेल सॉफ्टवेअर इंजिनिअरिंग संघाने परिभाषित केलेल्या फ्रेमवर्क क्रियाकलापांच्या संचामध्ये विभागले जाते. अँकर पॉइंट टप्पे - सर्पिलच्या मार्गावर प्राप्त झालेल्या कार्य उत्पादनांचे आणि परिस्थितींचे संयोजन - प्रत्येक उत्क्रांती पाससाठी नोंदवले जातात. स्पायरलभोवतीचा पहिला सर्किट उत्पादन तपशीलाच्या विकासात परिणाम करू शकतो; स्पायरलभोवतीचे त्यानंतरचे पास प्रोटोटाइप विकसित करण्यासाठी आणि नंतर सॉफ्टवेअरच्या अधिक अत्याधुनिक आवृत्त्या विकसित करण्यासाठी वापरले जाऊ शकतात.

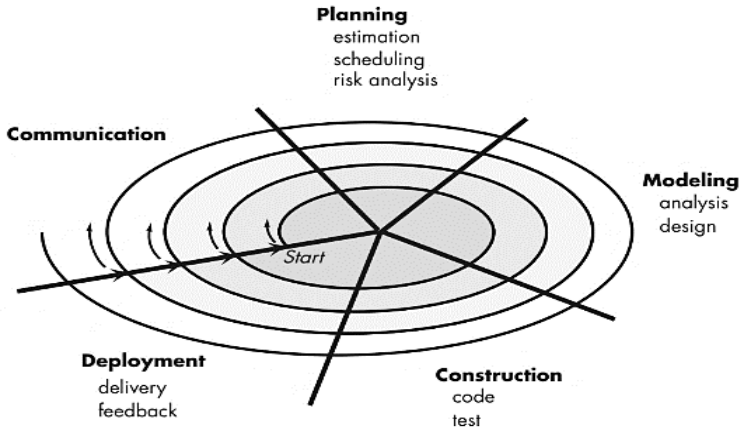


Fig. 1.8: स्पायरल मॉडेल (Spiral Model)

स्पायरल मॉडेलचे टप्पे :

1. **कम्युनिकेशन:** उपक्रम: भागधारकांशी कम्युनिकेशन
   उद्देश: रीक्वॅयरमेन्ट गोळा करण्यासाठी, अपेक्षा स्पष्ट करण्यासाठी आणि प्रगती प्रमाणित करण्यासाठी भागधारकांशी सतत कम्युनिकेशन राखणे.
2. **प्लॅनिंग :** उपक्रम: अंदाज, वेळापत्रक, रिस्क विश्लेषण

उद्देश: उद्दिष्टे परिभाषित करा, मर्यादा ओळखा, खर्चाचा अंदाज घ्या आणि रिस्क मूल्यांकन करा. स्पायरल मॉडेलची एक प्रमुख ताकद रिस्क व्यवस्थापनावर लवकर आणि वारंवार लक्ष केंद्रित करणे आहे.

3. **मॉडेलिंग:** उपक्रम: विश्लेषण, डिझाइन
   उद्देश: आवश्यकतांचे विश्लेषण केले जाते आणि सिस्टम आर्किटेक्चर डिझाइन केले जाते. प्रत्येक पुनरावृत्तीवर, नवीन वैशिष्ट्ये किंवा सुधारणा मॉडेल आणि नियोजित केल्या जातात.
4. **कन्स्ट्रक्शन:** उपक्रम: कोड, टेस्टिंग
   उद्देश: सॉफ्टवेअरचा विकास आणि टेस्ट वाढत्या प्रमाणात करा. प्रत्येक लूपच्या शेवटी एक कार्यरत उत्पादन (प्रोटोटाइप किंवा मॉड्यूल) वितरित केले जाते.
5. **डिप्लॉयमेंट :** उपक्रम: डिलिव्हरी , फीडबॅक
   उद्देश: विकसित आवृत्ती वापरकर्त्यांना किंवा भागधारकांना अभिप्रायासाठी वितरित केली जाते. अभिप्रायाच्या आधारे, पुढील पुनरावृत्तीमध्ये सुधारणा नियोजित केल्या जातात.

फायदे (Advantages) :

1. मजबूत रिस्क व्यवस्थापन.
2. मोठ्या आणि ध्येय-क्रिटिकल प्रकल्पांसाठी योग्य.
3. समस्या लवकर ओळखण्यास अनुमती देते.
4. संपूर्ण ग्राहकांच्या अभिप्रायांना प्रोत्साहन देते.

तोटे (Disadvantages) :

1. व्यवस्थापन आणि अंमलबजावणी करणे जटिल असू शकते.
2. वारंवार प्लॅनिंग आणि रिस्क विश्लेषणामुळे महाग.
3. लहान किंवा कमी रिस्क प्रकल्पांसाठी योग्य नाही.
4. रिस्क मूल्यांकनात उच्च-स्तरीय कौशल्य आवश्यक आहे.

1.7 एजाईल प्रोसेस मॉडेल: एक्सट्रीम प्रोग्रामिंग, ॲड़ाप्टिव्ह सॉफ्टवेअर डेव्हलपमेंट (ए एस डी), स्क्रॅम, डायनॅमिक सिस्टम डेव्हलपमेंट मेथड (डी एस डी एम), क्रिस्टल. एजाईल युनिफाइड प्रोसेस (ए यू पी)

(Agile Process Model: Extreme Programming, Adaptive Software Development (ASD), Scrum, Dynamic System Development Method (DSDM), CRYSTAL. Agile Unified Process (AUP))

1.7.1 ॲजाइल सॉफ्टवेअर डेव्हलपमेंट: एक्स्ट्रीम प्रोग्रामिंग (एक्स पी):

1. एक्सट्रीम प्रोग्रामिंग (XP) पुनरावृत्ती विकासासाठी 'अत्यंत' दृष्टिकोन घेते.
2. नवीन आवृत्त्या दिवसातून अनेक वेळा तयार केल्या जाऊ शकतात.
3. दर 2 आठवड्यांनी ग्राहकांना इनक्रिमेंट्स (Increments) दिली जातात.
4. प्रत्येक बिल्डसाठी सर्व चाचण्या केल्या पाहिजेत आणि चाचण्या यशस्वीरित्या चालल्या तरच बिल्ड स्वीकारले जाते.
5. एक्सट्रीम प्रोग्रामिंग रिलीज सायकल.

- **XP प्रोसेस:** एक्सट्रीम प्रोग्रामिंग त्याच्या पसंतीच्या विकास प्रतिमान म्हणून ऑब्जेक्ट-ओरिएंटेड दृष्टिकोनाचा वापर करते आणि चार फ्रेमवर्क क्रियाकलापांच्या संदर्भात होणारे नियम आणि पद्धतींचा संच समाविष्ट करते: प्लॅनिंग , डिझाइन, कोडिंग आणि टेस्टिंग .
- **XP - प्लॅनिंग :** कथांच्या संचाच्या निर्मितीपासून सुरुवात होते (ज्याला वापरकर्ता कथा देखील म्हणतात). प्रत्येक कथा ग्राहकाने लिहिली आहे आणि ती एका इंडेक्स कार्डवर ठेवली आहे. ग्राहक कथेला एक मूल्य (म्हणजे प्राधान्य) नियुक्त करतो. ॲजाइल टीम प्रत्येक कथेचे मूल्यांकन करते आणि किंमत नियुक्त करते.डिलिव्हरी करण्यायोग्य वाढीसाठी कथा गटबद्ध केल्या जातात. डिलिव्हरीच्या तारखेला एक वचनबद्धता केली जाते पहिल्या वाढीनंतर "प्रोजेक्ट वेग" इतर वाढींसाठी पुढील डिलिव्हरी तारखा निश्चित करण्यात मदत करण्यासाठी वापरला जातो.

- **XP - डिझाइन:** KIS (सोपे ठेवा) प्रिंसिपलचे पालन करते. CRC (वर्ग-जबाबदारी-सहयोगी) कार्ड्सचा वापर करण्यास प्रोत्साहन देते. कठीण डिझाइन समस्यांसाठी, "स्पाइक सोल्यूशन्स" तयार करण्याचे सुचवते—एक डिझाइन प्रोटोटाइप." रिफॅक्टरिंग" ला प्रोत्साहन देते—अंतर्गत प्रोग्राम डिझाइनचे पुनरावृत्ती परिष्करण. कोडिंग सुरू होण्यापूर्वी आणि नंतर डिझाइन दोन्ही होते
- **XP - कोडिंग:** कोडिंग सुरू होण्यापूर्वी प्रत्येक कथेसाठी युनिट चाचण्यांच्या मालिकेची रचना करण्याची शिफारस करते "पेअर प्रोग्रामिंग" ला प्रोत्साहन देते
    1. रिअल-टाईम समस्या सोडवणे आणि रिअल-टाईम क्वालिटी हमीसाठी यंत्रणा
    2. विकासकांना हातात असलेल्या समस्येवर लक्ष केंद्रित करते

  s/w च्या इतर भागांसह (कथा) सतत एकात्मता आवश्यक आहे, जे "स्मोक टेस्टिंग" वातावरण प्रदान करते
- **XP - टेस्टिंग:** टेस्टिंग स्वयंचलित करण्यासाठी फ्रेमवर्क वापरून युनिट चाचण्या अंमलात आणल्या पाहिजेत. हे रिग्रेशन टेस्टिंग धोरणाला प्रोत्साहन देते. एकत्रीकरण आणि प्रमाणीकरण टेस्टिंग दररोज होऊ शकते. स्वीकृती चाचण्या, ज्यांना ग्राहक चाचण्या देखील म्हणतात, ग्राहकाद्वारे निर्दिष्ट केल्या जातात आणि ग्राहकांच्या दृश्यमान कार्यक्षमतेचे मूल्यांकन करण्यासाठी अंमलात आणल्या जातात स्वीकृती चाचण्या वापरकर्त्याच्या कथांमधून घेतल्या जातात.

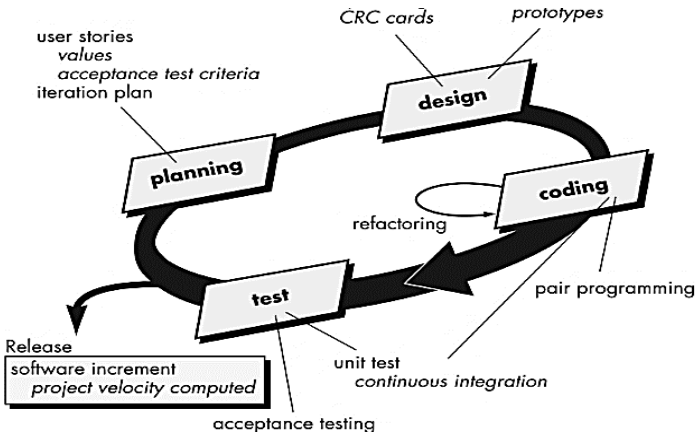


Fig. 1.9: एक्स्ट्रीम प्रोग्रॅमिंग (Extreme Programming)

1.7.2 अॅड़ाप्टिव्ह सॉफ्टवेअर डेव्हलपमेंट (ए एस डी) (Adaptive Software Development (ASD))

ही जटिल (complex) सॉफ्टवेअर आणि सिस्टम तयार करण्याची एक पद्धत आहे. एएसडी मानवी सहयोग आणि स्वयं-संघटनेवर लक्ष केंद्रित करते. एएसडी "लाइफ सायकल" मध्ये तीन टप्पे समाविष्ट आहेतः

1. अनुमान (Speculation)
2. सहयोग (Collaboration)
3. शिकणे (Learning)

हे खालीलप्रमाणे वर्णन केले आहे.

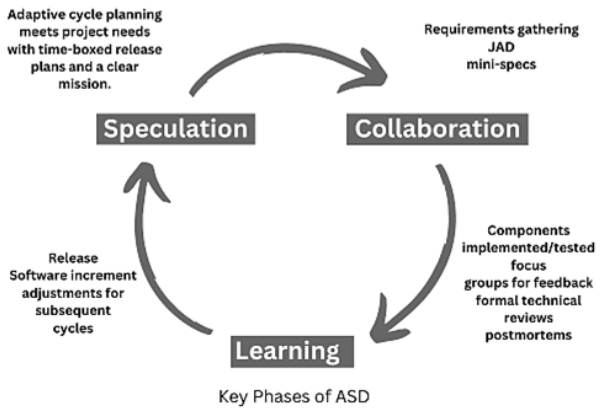


Fig. 1.10: अॅड़ाप्टिव्ह सॉफ्टवेअर डेव्हलपमेंट (ए एस डी) (Adaptive Software Development)

1. **अनुमान (Speculation) :** या टप्प्यात प्रकल्प सुरू केला जातो आणि नियोजन केले जाते. प्रोजेक्ट प्लॅन प्रोजेक्टची आवश्यकता, वापरकर्त्याच्या गरजा, ग्राहक मिशन स्टेटमेंट इ. सारख्या प्रकल्पाची दीक्षा माहिती वापरते, प्रकल्प इच्छित असलेल्या रिलीझ चक्रांचा संच परिभाषित करण्यासाठी.
2. **सहयोग (Collaboration):** हे संप्रेषण आणि कार्यसंघ सहकार्य करते परंतु वैयक्तिकृततेवर जोर देते कारण सर्जनशील विचारात वैयक्तिक सर्जनशीलता मोठी भूमिका बजावते.
3. **शिकणे (Learning):** प्रत्येक विकास चक्र म्हणजे शिकण्याची आणि परिस्थितीशी जुळवून घेण्याची संधी असतेसंघ प्रत्येक पुनरावृत्तीच्या निकालांचे विश्लेषण करून त्यांची समज सुधारतात आणि भविष्यातील योजना . .समायोजित करतात

1.7.3 स्क्रॅम (Scrum)

आहे जी पुनरावृत्ती आणि इनक्रिमेन्ट दृष्टिकोन वापरून सॉफ्टवेअर विकास व्यवस्थापित आणि नियंत्रित करण्यासाठी वापरली जाऊ शकते. येथे हलकी संज्ञा म्हणजे उत्पादक वेळ जास्तीत जास्त करण्यासाठी प्रक्रियेचा ओव्हरहेड शक्य तितका लहान ठेवला जातो. स्क्रम प्रक्रियेचा एकूण प्रवाह आकृतीमध्ये दर्शविला आहे. स्क्रम सॉफ्टवेअर प्रोसेस नमुन्यांच्या संचाच्या वापरावर भर देतो जे घट्ट टाईमलाइन, बदलत्या आवश्यकता आणि बीसनेस गंभीरता असलेल्या प्रकल्पांसाठी प्रभावी सिद्ध झाले आहेत. या प्रत्येक प्रोसेस नमुन्यात विकास कृतींचा संच परिभाषित केला जातो:

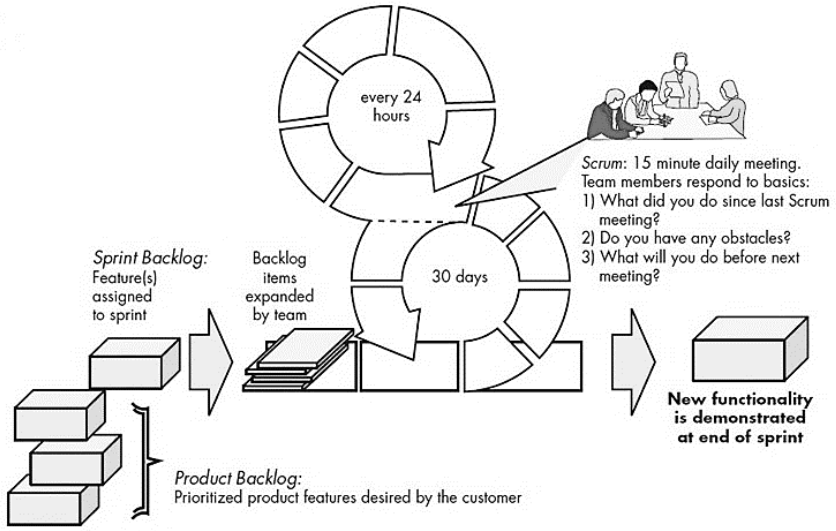


Fig. 1.11: स्क्रम (Scrum)

बॅकलॉग- प्रोजेक्ट आवश्यकतांची किंवा वैशिष्ट्यांची प्राधान्यीकृत यादी जी ग्राहकांसाठी बीसनेस मूल्य प्रदान करते. आयटम कधीही बॅकलॉगमध्ये जोडल्या जाऊ शकतात (अशा प्रकारे बदल सादर केले जातात). उत्पादन व्यवस्थापक बॅकलॉगचे मूल्यांकन करतो आणि आवश्यकतेनुसार प्राधान्यक्रम अद्यतनित करतो. स्प्रिंट्स- मध्ये अशा कामाच्या युनिट्स असतात ज्या बॅकलॉगमध्ये परिभाषित केलेली आवश्यकता साध्य करण्यासाठी आवश्यक असतात जी पूर्वनिर्धारित टाईम-बॉक्समध्ये (सामान्यत: 30 दिवस) बसली पाहिजेत. स्प्रिंट दरम्यान बदल (उदा., बॅकलॉग कामाचे आयटम) सादर केले जात नाहीत. म्हणून, स्प्रिंट टीम सदस्यांना अल्पकालीन परंतु स्थिर वातावरणात काम करण्यास अनुमती देते.स्क्रम मीटिंग्ज - स्क्रम टीमद्वारे दररोज आयोजित केलेल्या लहान (सामान्यत: 15 मिनिटांच्या) बैठका असतात.

सर्व टीम सदस्यांकडून तीन प्रमुख प्रश्न विचारले जातात आणि त्यांची उत्तरे दिली जातात:

- शेवटच्या टीम मीटिंगपासून तुम्ही काय केले?
- तुम्हाला कोणते अडथळे येत आहेत?
- पुढील टीम मीटिंगद्वारे तुम्ही काय साध्य करण्याचा प्लॅन आखली आहे?

स्क्रम मास्टर नावाचा एक टीम लीडर मीटिंगचे नेतृत्व करतो आणि प्रत्येक व्यक्तीच्या प्रतिसादांचे मूल्यांकन करतो. स्क्रम मीटिंग टीमला शक्य तितक्या लवकर संभाव्य समस्या उघड करण्यास मदत करते. तसेच, या दैनंदिन बैठकींमुळे "ज्ञानाचे सामाजिकीकरण" होते आणि त्याद्वारे एक स्वयं-संघटनात्मक संघ रचना निर्माण होते जी ग्राहकांना सॉफ्टवेअर वाढीचा लाभ देते जेणेकरून ती कार्यक्षमता अंमलात आणली गेली आहे. डेमो - ग्राहकांना सॉफ्टवेअर इनक्रिमेन्ट प्रदान करणे

जेणेकरून अंमलात आणलेली कार्यक्षमता ग्राहकाद्वारे प्रदर्शित आणि मूल्यांकन केली जाऊ शकेल. हे लक्षात ठेवणे महत्त्वाचे आहे की डेमोमध्ये सर्व नियोजित कार्यक्षमता असू शकत नाही, परंतु त्या कार्ये असू शकतात जी स्थापित केलेल्या वेळेच्या चौकटीत वितरित केली जाऊ शकतात.

फायदे (Advantages) :

1. स्क्रम मॉडेल प्रोजेक्ट विकास स्थितीत पारदर्शकता आणते.
2. ते बदलांसाठी लवचिकता प्रदान करते.
3. सुधारित संवाद आहे, विकास प्रक्रियेत किमान ओव्हरहेड आहे.
4. उत्पादकता सुधारली जाऊ शकते.

तोटे (Disadvantages) :

1. काही निर्णय निश्चित कालावधीत ट्रॅक करणे कठीण असते.
2. सिस्टिम च्या गैर-कार्यक्षम आवश्यकतांना सामोरे जाण्यासाठी समस्या आहेत.

1.7.4 डायनॅमिक सिस्टम्स डेव्हलपमेंट टेक्निक (डी एस डी एम) (The Dynamic Systems Development technique (DSDM)):

एक असोसिएट डिग्री एजाईल कोड विकास दृष्टीकोन आहे जो प्रणाली तयार करणे आणि देखरेख करण्यासाठी एक फ्रेमवर्क प्रदान करतो. डीएसडीएम तत्त्वज्ञान समाजशास्त्रज्ञ तत्त्वाच्या सुधारित आवृत्तीमधून घेतले जाते - अनुप्रयोगाच्या 80 % बहुतेक वेळा वीस टक्के वेळेत वितरित केले जाते ज्याच्या इच्छेनुसार संपूर्ण (100 टक्के) अनुप्रयोग वितरित केला जातो. डीएसडीएम ही एक पुनरावृत्ती कोड पद्धत आहे ज्यामध्ये प्रत्येक पुनरावृत्ती 80% नियम पाळते जी प्रत्येक वाढीसाठी खालील वाढीसाठी हालचाल सुलभ करण्यासाठी फक्त पुरेसे काम आवश्यक असते. एकदा बऱ्याच व्यवसायांची आवश्यकता लक्षात घेतल्यास किंवा बदलांची विनंती केली आणि सामावून घेतल्यानंतर उर्वरित तपशील बऱ्याचदा नंतर पूर्ण केला जातो.

डी एस डी एम चे टप्पे: (Phases of DSDM)

1. **पूर्व-प्रकल्प (Pre-project):** प्रकल्प व्यवहार्य आहे आणि व्यवसाय लक्ष्यांसह संरेखित आहे याची खात्री देते.
2. **व्यवहार्यता अभ्यास (Feasibility Study):** प्रकल्प तांत्रिक आणि आर्थिकदृष्ट्या व्यवहार्य आहे की नाही हे निर्धारित करते.
3. **फाउंडेशन (Foundations):** हा टप्पा प्रकल्पाची व्याप्ती, बजेट आणि टाईमलाइन स्थापित करतो आणि प्रारंभिक प्रकल्प योजना परिभाषित करतो.
4. **अन्वेषण (Exploration):** सिस्टमचे मॉडेल आणि फंक्शनल प्रोटोटाइप विकसित करते.
5. **अभियांत्रिकी (Engineering):** प्रोटोटाइप एक मजबूत, तैनात करण्यायोग्य सोल्यूशनमध्ये वळा.
6. **वाढीव उपयोजन (Incremental Deployment):** कार्यरत सोल्यूशनमध्ये प्रोटोटाइप विकसित करते आणि थेट वातावरणात वाढीमध्ये समाधान तैनात करते.
7. **पोस्ट-प्रोजेक्ट (Post-project):** दीर्घकालीन प्रभावीपणा आणि देखभाल सुनिश्चित करण्यासाठी पुनरावलोकने आयोजित करते.

आकृती खाली डी एस डी एम जीवन चक्र वर्णन करते:

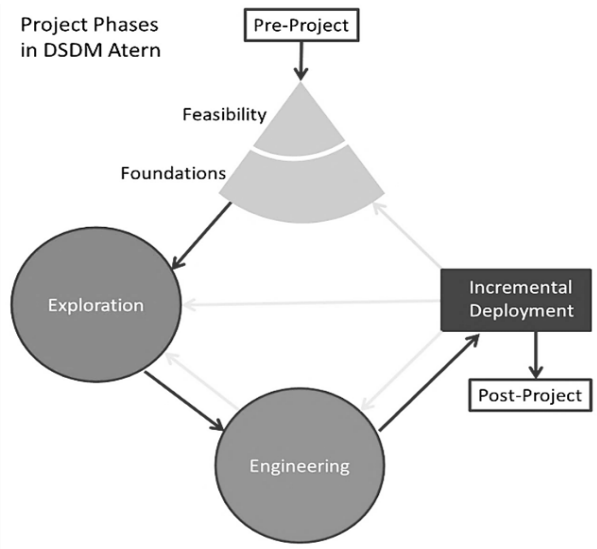


Fig. 1.12: डायनॅमिक सिस्टम्स डेव्हलपमेंट टेक्निक (डी एस डी एम) (Dynamic Systems Development technique (DSDM))

1.7.5 क्रिस्टल (Crystal):

एजाईल विकास/फ्रेमवर्कमध्ये क्रिस्टल पद्धती क्रिस्टल पद्धत एक एजाईल फ्रेमवर्क आहे जी एक हलकी किंवा एजाईल पद्धती मानली जाते जी व्यक्ती आणि त्यांच्या परस्परसंवादावर लक्ष केंद्रित करते. पद्धती मानवी जीवनासाठी महत्त्वपूर्ण जोखमीसाठी रंग-कोडित आहेत. हे मुख्यतः एकाच कार्यक्षेत्रातून काम करणाऱ्या विकसकांच्या टीमद्वारे अल्प-मुदतीच्या प्रकल्पांसाठी आहे. काही एजाईल सॉफ्टवेअर डेव्हलपमेंट लाइफ सायकल (एसडीएलसी) मॉडेल्समध्ये क्रिस्टलला एजाईल एसडीएलसी मॉडेलपैकी एक मानले जाते.

क्रिस्टल पद्धतीची दोन मुख्य श्रद्धा:

- वर्कफ्लो ऑप्टिमाइझ करण्यासाठी आपला स्वतःचा मार्ग आणि पद्धती शोधा.
- प्रकल्प अद्वितीय आणि डायनॅमिक बनविण्यासाठी अद्वितीय पद्धतींचा वापर करा.

क्रिस्टलची मुख्य तत्त्वे:

- वापरण्यायोग्य कोडची वारंवार वितरण.
- पूर्वसूचनांद्वारे प्रतिबिंबित सुधारणा.
- ऑस्मोटिक कम्युनिकेशन - सामायिक जागेत जवळून कार्य करणे.
- वैयक्तिक सुरक्षा - कार्यसंघ सदस्यांनी बोलणे सुरक्षित वाटले पाहिजे.
- फोकस - वारंवार व्यत्यय न घेता लक्ष केंद्रित करण्याची क्षमता.
- तांत्रिक वातावरण - स्वयंचलित चाचणी, आवृत्ती नियंत्रण, इ. चे समर्थन करणारी साधने वापरा.

फायदे (Advantages) :

1. विशिष्ट कार्यसंघाच्या गरजा अनुरूप.
2. संप्रेषण आणि सहकार्यास प्रोत्साहित करते.
3. कमी ओव्हरहेड आणि दस्तऐवजीकरण.
4. लवचिक आणि अनुकूली.

तोटे (Disadvantages) :

1. कमी संरचित, अत्यधिक नियमन केलेल्या वातावरणात चांगले कार्य करू शकत नाही.
2. अनुभवी आणि शिस्तबद्ध संघांची आवश्यकता आहे.
3. रुपांतर न करता खूप मोठ्या किंवा मिशन-क्रिटिकल प्रकल्पांसाठी योग्य नाही.

1.7.6 एजाईल युनिफाइड प्रोसेस (एयूपी) (Agile Unified Process (AUP))

एजाईल युनिफाइड प्रोसेस (ए यू पी) स्कॉट एम्बलरने विकसित केलेल्या तर्कसंगत युनिफाइड प्रोसेस (आरयूपी) ची एक सरलीकृत आवृत्ती आहे. हे सॉफ्टवेअर डेव्हलपमेंट लाइफसायकल (एसडीएलसी) मध्ये एजाईल तत्त्वे समाविष्ट करते,

ज्यामुळे ते फिकट, वापरण्यास सुलभ आणि अधिक अनुकूलन करते. एयूपी सतत चाचणीस प्रोत्साहित करते आणि विकासाच्या प्रत्येक टप्प्यात त्यास समाकलित करते.

ए यू पी ची वैशिष्ट्ये (Features) :

- **पुनरावृत्ती आणि वाढीव (Iterative and Incremental):** एयूपी लहान चक्रात कार्यरत सॉफ्टवेअर वितरित करण्यावर जोर देते, लवचिकता आणि बदलत्या आवश्यकतांमध्ये रुपांतर करण्यास अनुमती देते.
- **सहकार्यावर लक्ष केंद्रित करा (Focus on Collaboration):** एयूपी विकसक, भागधारक आणि अंत-वापरकर्त्यांमधील सहकार्यास प्राधान्य देते, सामायिक समज आणि सतत अभिप्राय वाढवते.
- **किमान मॉडेलिंग (Minimalist Modeling):** एयूपी तपशीलवार कागदपत्रांऐवजी विकासास समर्थन देण्यासाठी फक्त पुरेसे मॉडेल तयार करण्यावर लक्ष केंद्रित करून हलके मॉडेलिंग दृष्टिकोनास प्रोत्साहित करते.
- **सात विषय (Seven Disciplines):** आरयूपीच्या विपरीत, एयूपीकडे सात मुख्य विषय आहेत: मॉडेलिंग, अंमलबजावणी, चाचणी, उपयोजन, कॉन्फिगरेशन व्यवस्थापन, प्रकल्प व्यवस्थापन आणि पर्यावरण.
- **साधन स्वातंत्र्य (Tool Independence):** एयूपी कार्यसंघांना विशिष्ट साधनांचा वापर करण्यास भाग पाडण्याऐवजी त्यांच्या गरजा भागविणारी साधने निवडण्याची परवानगी देते.
- **एजाईल तत्त्वे (Agile Principles):** एयूपी मूल्य, सहकार्य आणि लवचिकतेवर लक्ष केंद्रित करून एजाईल युतीच्या तत्त्वांचे पालन करते.
- **ए यू पी चे टप्पे (Phases of AUP):** एयूपी चार-फेज लाइफसायकलचे अनुसरण करते: स्थापना, विस्तार, बांधकाम आणि संक्रमण.
  1. **स्थापना (Inception):** हा टप्पा प्रकल्प व्याप्ती, व्यवहार्यता आणि प्रारंभिक आवश्यकता परिभाषित करण्यावर लक्ष केंद्रित करतो.
  2. **विस्तार (Elaboration):** आवश्यकता पुढील परिष्कृत केल्या आहेत आणि तपशीलवार आर्किटेक्चर विकसित केले आहे.
  3. **कॅन्स्ट्रक्शन (Construction):** सॉफ्टवेअर तयार आणि पुनरावृत्ती चक्रात चाचणी केली जाते.
  4. **संक्रमण (Transition):** अंतिम उत्पादन तैनात केले आहे आणि प्रकल्प बंद आहे.

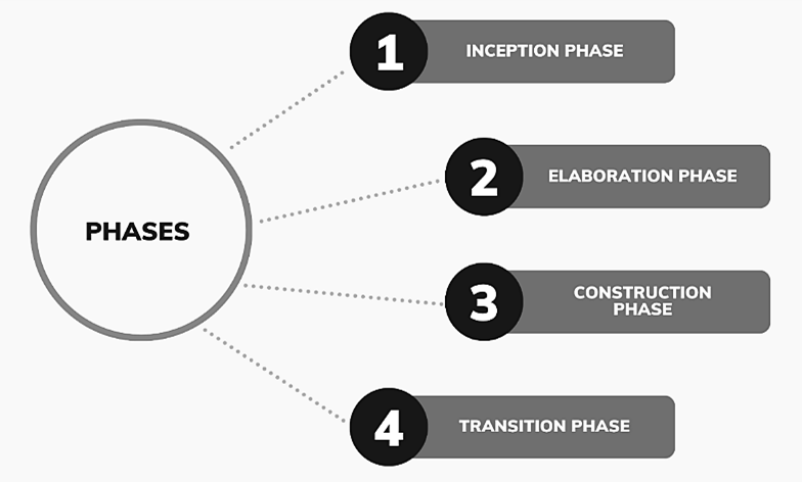


Fig. 1.13: फेसेस ऑफ एजाईल युनिफाईड प्रोसेस (Phases of Agile Unified Process)

References:

1. Software Engineering: A practitioner's approach by Roger S. Pressman & Bruce R. Maxim, McGraw Hill Higher Education, New Delhi, (Ninth Edition) ISBN 93-5532-504-5
2. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
3. Software Engineering: Principles and practices by Deepak Jain, Oxford University Press, New Delhi ISBN 9780195694840

4. Software Testing: Principles and Practices by Srinivasan Desikan, Gopalaswamy Ramesh, PEARSON Publisher: Pearson India 2007, ISBN: 978-81-7758-121-8
5. Software Testing by Ron Patton, Sams Publishing; 2nd edition, 2005 ISBN: 0672327988

युनिट-2
सॉफ्टवेअर रिक्वारमेन्ट इंजिनिअरिंग
(Software Requirement Process)

विषय निष्पत्ती (Course Outcome):

CO2: सॉफ्टवेअर रिक्वारमेंट स्पेसिफिकेशन (requirement specification) तयार करा.

घटक निष्पत्ती (Theory Learning Outcome - TLO):

1. दिलेल्या समस्येसाठी सॉफ्टवेअर अभियांत्रिकीची तत्त्वे लागू करा.
2. दिलेल्या समस्येसाठी संबंधित आवश्यकता (requirement) अभियांत्रिकी चरण (steps) निवडा.
3. दिलेल्या समस्येसाठी आवश्यक (requirement) अभियांत्रिकी मॉडेल (model) तयार करा.
4. दिलेल्या समस्येसाठी एसआरएस (SRS) तयार करा

2.1 सॉफ्टवेअर अभियांत्रिकी कोअर तत्त्वे (Software Engineering Core Principle) :

तत्त्व (Principle) म्हणजे विचारांच्या प्रणालीमध्ये आवश्यक एक महत्त्वाचा कायदा .
ठोस सॉफ्टवेअर अभियांत्रिकी अभ्यासासाठी काही मानके किंवा नियम स्थापित करण्यात तत्त्व आम्हाला मदत करतात. डेव्हिड हूकरने (David Hooker) खाली सूचीबद्ध केल्यानुसार सॉफ्टवेअर अभियांत्रिकी अभ्यासावर लक्ष केंद्रित करणारे सात मुख्य तत्त्वे प्रस्तावित केली आहेत.

1. **आपले मूल्य प्रदान करा (हे सर्व अस्तित्त्वात आहे त्याचे कारण)** : **(Provide value to yours (The reason it all exists)):** वैशिष्ट्ये (specification) तयार करण्यापूर्वी आणि हार्डवेअर प्लॅटफॉर्म किंवा विकास प्रोसेस (development process) निश्चित करण्यापूर्वी, हे सिस्टममध्ये वास्तविक मूल्य जोडते याची खात्री करा. जर ते तसे करत नसेल तर सॉफ्टवेअर सिस्टम केवळ एका कारणास्तव अस्तित्त्वात आहे म्हणजेच त्याच्या वापरकर्त्यांना मूल्य (value) प्रदान करण्यासाठी.
2. **कीप इट सिम्पल स्टुपिड (Keep it Simple, Stupid (KISS)):** सॉफ्टवेअर डिझाइन (software design) शक्य तितके सोपे असावे. पण सोपे म्हणजे कोणतेही अंतर्गत फिचर्स (internal features) काढून टाकणे किंवा 'घाईघाईने आणि कसेतरी केलेले' असे नाही. सोपे म्हणजे - सॉफ्टवेअरमध्ये (software) त्रुटी कमी असणे, समजण्यास सोपे असणे आणि देखभाल करण्यास सोपे असणे.
3. **दूरदृष्टी राखून ठेवा (Maintain the vision) :** सॉफ्टवेअर प्रकल्पाच्या यशासाठी स्पष्ट दृष्टी आणि वैचारिक अखंडता आवश्यक आहे.
4. **आपण काय तयार करता त्याचा वापर इतर करतील (What you produce, others will consume) :** सॉफ्टवेअर इंजिनियरने नेहमी हे जाणून सॉफ्टवेअर प्रॉडक्ट (software product) स्पेसिफाय (specify), डिझाइन (design) आणि इंप्लिमेंट (implement) केले पाहिजे की, दुसरे कोणीतरी तुम्ही काय करत आहात हे समजून घेणार आहे.. त्याच्या उत्पादनाचे वापरकर्ते संभाव्यत: मोठे (potentially large) आहेत. शेवटच्या वापरकर्त्याचे (end users) लक्षात ठेवून त्याने डिझाइन आणि कोड डिझाइन केले पाहिजे कारण शेवटच्या वापरकर्त्याचे कार्य सुलभ केल्याने सिस्टमला मूल्य (value of the system) वाढते.
5. **भविष्यासाठी खुले करा (Be open to future) :** दीर्घ आयुष्यासह प्रणालीचे (system) अधिक मूल्य (value) असते. अशाप्रकारे, एक प्रणाली (system) डेव्हलप केली जावी जेणेकरून संपूर्ण प्रणाली पुन्हा वापरली जाऊ शकते आणि जर ते बदलांशी जुळवून घेण्यायोग्य असेल तरच हे शक्य आहे.
6. **पुनर्वापरासाठी पुढे प्लॅन: (Plan ahead for reuse) :** पुनर्वापर वेळ आणि मेहनत वाचवतो. ऑब्जेक्ट ओरिएंटेड टेक्नॉलॉजीजचा (object oriented technologies) वापर कोड (code) आणि डिझाइनच्या (design) पुनर्वापराचा फायदा होतो. पुनर्वापर किंमत (value) आणि पुन्हा वापरण्यायोग्य घटक आणि ज्या सिस्टममध्ये ते समाविष्ट केले गेले आहेत त्यांचे मूल्य (value) वाढवते.

7. **करण्यापूर्वी विचार करा (Think before you do) :** कोणतेही काम करण्यापूर्वी नेहमीच स्पष्ट आणि पूर्ण नियोजन करा. हे चांगले परिणाम (result) देईल. जेव्हा आपण एखाद्या गोष्टीबद्दल विचार करता तेव्हा आपण हे योग्य करण्याची अधिक शक्यता असते किंवा पुढच्या वेळी हे कसे करावे याबद्दल आपण कमीतकमी ज्ञान मिळवू शकता.

2.2 सॉफ्टवेअर पद्धती (Software Practices) : सॉफ्टवेअर अभियांत्रिकी (Software engineering) पद्धतींमध्ये सॉफ्टवेअर सिस्टम (software system) प्रभावीपणे डिझाइन (design), विकसित (develop) करण्यासाठी आणि देखरेखीसाठी वापरल्या जाणाऱ्या तत्त्वे, पद्धती आणि प्रोसेस समाविष्ट आहेत. हे खालीलप्रमाणे आहेत:

2.2.1 संप्रेषण (Communication) :

संप्रेषण (Communication) पद्धतींमध्ये प्रारंभिक अवस्था म्हणजे ग्राहकांकडून आवश्यकता माहिती गोळा करणे.

संप्रेषणाची तत्त्वे खालीलप्रमाणे आहेत:

1. **काळजीपूर्वक ऐकणे (Listen carefully) :** ग्राहकांच्या आवश्यकता काळजीपूर्वक आणि उत्तम प्रकारे समजण्यासाठी. ऐकण्याचे तत्व (listening principle) स्पीकरकडून योग्य माहिती संकलन (data collection) सुनिश्चित करते. मनामधे काय शंका असेल तर त्याचे निरसन करून घेतले पाहिजे. दरम्यान व्यत्यय (interruption) टाळण्याचा प्रयत्न करा. हे चर्चेत युक्तिवाद (arguments) करू नये.
2. **संप्रेषणापूर्वीचे निवारण (Preparation before communication) :** बैठकीसाठी अजेंडा (agenda) तयार करणे फार महत्वाचे आहे. अजेंड्यामध्ये मीटिंगमध्ये चर्चा करायच्या मुद्द्यांचा समावेश असतो.
3. **तयार केलेला क्रियाकलाप सुलभ करा (Be prepared facilitate the activity) :**
   बैठक आयोजित करण्यासाठी आणि अजेंडा सेट करण्यासाठी फॅसिलिटेटर (facilitator) असावा.
4. **संवादास सामोरे जाण्याचा प्रयत्न करा (Face to face communication) :**
   चर्चेवर लक्ष केंद्रित करण्यासाठी काही दस्तऐवजीकरण (documentation) किंवा सादरीकरणासह (presentation) सज्ज व्हा.
5. **नोट्स घ्या आणि निर्णयाचे दस्तऐवजीकरण करा (Take notes and document the decision) :**
   बैठकीच्या चर्चेत उपस्थित केलेले सर्व महत्त्वाचे मुद्दे आणि अडचणी विषयीचे मुद्दे लिहा.
6. **सहकार्याचा प्रयत्न करा (Try for collaboration) :** सर्वोत्कृष्ट कार्यसंघासाठी कार्यसंघ सदस्यांमधील सहकार्य (collaboration) खूप महत्वाचे आहे. जेव्हा कार्यसंघाच्या सदस्यांचे सामूहिक ज्ञान सिस्टम फंक्शन्सचे (system function) उत्पादन स्पष्ट करण्यासाठी एकत्रित होते तेव्हा हे साध्य होते.
7. **लक्ष केंद्रित करा, मॉड्यूलराइझ चर्चा (Be focused, modularize discussion) :** कोणत्याही प्रकारच्या **संवादात** (communication) अनेक लोक सामील असतात आणि चर्चा एका विषयावरून दुसऱ्या विषयावर जात राहते. फॅसिलिटेटरने (facilitator) संभाषण मॉड्यूलर (modular) ठेवले पाहिजे, एक विषय पूर्ण झाल्यावरच तो सोडून द्यावा.
8. **काहीतरी अस्पष्ट असल्यास चित्र काढा: (Draw a picture if something is unclear) मौखिक संवाद** (Verbal Communication) फक्त या तत्त्वावर चालतो. जेव्हा शब्द काम करत नाहीत (अयशस्वी होतात), तेव्हा **चित्र** (drawing) अनेकदा स्पष्टता (clarity) प्रदान करू शकते.
9. **(अ) एखाद्या गोष्टीस सहमती द्या, पुढे जा** (Agree to something, move on)
   (ब) एखाद्या गोष्टीस सहमत होऊ शकत नाही, पुढे जा (Can not agree to something, move on)
   (क) एखादे कार्य अस्पष्ट असल्यास आणि याक्षणी स्पष्टीकरण दिले जाऊ शकत नसल्यास, पुढे जा
   (if a function is unclear and can not be clarified at the moment, move on) :
   जे लोक संभाषणात भाग घेतात त्यांनी हे ओळखले पाहिजे की बऱ्याच विषयांवर चर्चेची आवश्यकता असते आणि कधीकधी संभाषण एजाईल ता (communication agility) मिळविण्याचा उत्तम मार्ग म्हणजे "पुढे जाणे" (moving on).
10. **निगोशिएशन ही स्पर्धा नाही (Negotiation is not a contest) :**अनेक परिस्थितींमध्ये, सॉफ्टवेअर डेव्हलपर आणि कस्टमरला फंक्शन्स, प्राधान्यक्रम (priorities) आणि फीचर्स तसेच वितरण तारखांवर (delivery dates) निगोशिएट करावे लागते. जर सॉफ्टवेअर प्रोजेक्ट टीमने चांगले कोलॅबोरेट (collaborated)

केले असेल, तर सर्व पक्षांचे एक समान ध्येय किंवा उद्दिष्टे (common goal or objectives) असतात, यामुळे निगोशिएशनमध्ये सर्व पक्षांकडून तडजोड (compromise) करावी लागेल.

2.2.2 प्लॅनिंग (Planning) :

नियोजन क्रियाकलापात (planning activity) तांत्रिक आणि व्यवस्थापन पद्धतींचा एक संच असतो. तांत्रिक आणि व्यवस्थापन पद्धती ज्या सॉफ्टवेअर प्रोजेक्ट (software project) टीमला रस्ता नकाशाची व्याख्या करण्यास सक्षम करतात कारण ती त्याच्या धोरणात्मक उद्दीष्टे आणि ध्येयांकडे **(**strategic objectives and goal**)** प्रवास करते.

नियोजन क्रियाकलापांमध्ये खालील तत्त्वे असतात:

1. **प्रकल्पाची व्याप्ती समजून घेण्यासाठी (To understand the scope of the project) :** प्रोजेक्टची व्याप्ती सॉफ्टवेअर प्रोजेक्ट (software project) टीमला गंतव्यस्थान (destination) प्रदान करते.
2. **ग्राहकांना नियोजन क्रियाकलापात सामील करण्यासाठी (To involve the customer in the planning activity) :** ग्राहक सॉफ्टवेअर प्रकल्प (software project) प्राधान्यक्रम (Priorities) परिभाषित करतो आणि प्रकल्प अडचणी स्थापित करतो. हे कार्य साध्य करण्यासाठी सॉफ्टवेअर अभियंत्यांनी बऱ्याचदा टाईमलाइन (timeline)**,** ऑर्डर वितरण (order delivery**)** आणि इतर प्रकल्प संबंधित समस्यांशी बोलणी करणे आवश्यक आहे.
3. **नियोजन पुनरावृत्ती आहे हे ओळखण्यासाठी (To recognize that planning is iterative) :** सॉफ्टवेअर प्रकल्प (software project) योजना कधीही दगडात कोरली जात नाही. सॉफ्टवेअर प्रोजेक्टचे कार्य सुरू होत असताना**,** गोष्टी बदलण्याची शक्यता आहे. अशा प्रकारे या बदलांना सामावून घेण्यासाठी योजना समायोजित करणे आवश्यक आहे. याव्यतिरिक्त**,** पुनरावृत्ती**,** वाढीव प्रोसेस मॉडेल (model) ग्राहकांकडून प्राप्त झालेल्या अभिप्रायाच्या आधारे रिप्लॅनिंगचे (replanning) आदेश देतात.
4. **आपल्या माहितीवर आधारित अंदाज करणे (To estimate based on what you know) :** सॉफ्टवेअर प्रोजेक्ट (software project) टीमच्या कामाच्या सध्याच्या समजुतीवर आधारित खर्च**,** प्रयत्न आणि कार्य कालावधीचे संकेत देणे हे अंदाजाचे उद्दीष्ट आहे. जर माहिती अविश्वसनीय असेल तर अंदाज तितकाच अविश्वसनीय असेल.
5. **आपण योजना परिभाषित करता तेव्हा जोखमीचा विचार करणे (To consider risk as you define the plan) :** जर सॉफ्टवेअर प्रोजेक्ट (software project) टीमने उच्च संभाव्यता आणि प्रभाव असलेल्या जोखमीची व्याख्या केली असेल तर आकस्मिक नियोजन आवश्यक आहे. यापैकी एक किंवा अधिक जोखीम येण्याची शक्यता सामावून घेण्यासाठी सॉफ्टवेअर प्रकल्प (software project) योजना समायोजित केली जावी**.**
6. **वास्तववादी व्हा: (Be realistic)** सॉफ्टवेअर प्रोजेक्ट software project टीमचा प्रत्येक सदस्य दररोज 100 टक्के कार्य करत नाही. तर**,** प्रत्येक व्यक्तीद्वारे दररोज किती केले जाऊ शकते आणि ते किती चांगले केले जाऊ शकते यावर वास्तववादी व्हा.
7. **आपण योजना परिभाषित करता तेव्हा ग्रॅन्युलॅरिटी समायोजित करण्यासाठी (To adjust granularity as you define the plan) :** ग्रॅन्युलॅरिटी या शब्दाचा अर्थ**,** जेव्हा एखादा प्रोजेक्ट प्लॅन विकसित केला जातो किंवा **इंजिनियर्ड** (engineered) केला जातो**,** तेव्हा त्यात समाविष्ट केलेल्या तपशिलाची पातळी (level of detail) होय.
   - **फाइन ग्रॅन्युलॅरिटी प्लॅन (Fine Granularity Plan):** या प्लॅनमध्ये कामाच्या कार्यांचे महत्त्वपूर्ण तपशील (significant detail) असतात, जे तुलनेने कमी वेळेच्या वाढीवर (short time increments) नियोजित केले जातात.
   - **कोअर्स ग्रॅन्युलॅरिटी प्लॅन (Coarse Granularity Plan):** हा प्लॅन कामाच्या कार्यांचे (work tasks) विस्तृत तपशील (broader details) प्रदान करतो, जे जास्त कालावधीसाठी (longer time periods) नियोजित केले जातात.

   मूलभूतपणे, प्रोजेक्टची टाईमलाइन सध्याच्या तारखेपासून पुढे सरकते तसतशी ग्रॅन्युलॅरिटी फाइनपासून कोअर्सकडे (coarse) जाते**.** पुढील काही आठवड्यांसाठी किंवा महिन्यांसाठी**,** सॉफ्टवेअर प्रोजेक्टचे महत्त्वपूर्ण तपशिलात प्लॅनिंग केले जाऊ शकते**.**

8. **गुणवत्ता सुनिश्चित करण्याचा आपला कसा हेतू आहे हे परिभाषित करण्यासाठी (To define how you intend to ensure quality) :** सॉफ्टवेअर प्लॅनने हे ओळखले पाहिजे की सॉफ्टवेअर प्रोजेक्ट टीम गुणवत्ता (quality) कशी सुनिश्चित करते.या तत्त्वासाठी (principle) दोन तंत्रे (techniques) वापरली जातात:
    1. फॉर्मल टेक्निकल रिव्ह्यू (formal technical review)
    2. पेअर प्रोग्रामिंग (pair programming)

   जेव्हा फॉर्मल टेक्निकल रिव्ह्यूज (formal technical reviews) वापरले जातात, तेव्हा ते नियोजित (scheduled) असावेत. जेव्हा कंस्ट्रक्शन दरम्यान पेअर प्रोग्रामिंग वापरले जाते, तेव्हा ते प्लॅनमध्ये स्पष्टपणे परिभाषित (explicitly defined) केले पाहिजे.
9. **आपण बदलास सामावून घेण्याचा आपला हेतू कसा आहे हे वर्णन करण्यासाठी: (To describe how you intend to accommodate change)** अनियंत्रित बदलांद्वारे सर्वोत्कृष्ट नियोजन केले जाऊ शकते. सॉफ्टवेअर प्रोजेक्ट (software project) टीमने सॉफ्टवेअर अभियांत्रिकी (software engineering) कार्य चालू असताना बदल कसे सामावून घ्यावे हे ओळखले पाहिजे.
10. **वारंवार योजनेचा मागोवा घेणे आणि आवश्यकतेनुसार समायोजन करणे: (To track the plan frequently and to make adjustments as required)** अनेक सॉफ्टवेअर प्रोजेक्ट्स दररोज थोडे थोडे करत वेळापत्रकापेक्षा (schedule) मागे पडतात. म्हणूनच, प्रगतीचा (progress) दररोज मागोवा घेणे, समस्या असलेली क्षेत्रे (problem areas) आणि नियोजित काम (scheduled work) प्रत्यक्षात केलेल्या कामाशी (actual work conducted) जुळत नसलेल्या परिस्थिती शोधणे योग्य ठरते.

2.2.3 मॉडेलिंग (Modelling) :

तांत्रिक स्तरावर, सॉफ्टवेअर अभियांत्रिकी मॉडेलिंग (software engineering modeling) कार्यांच्या **सिरीज**पासून सुरू होते ज्यामुळे आवश्यकतेचे संपूर्ण तपशील आणि सॉफ्टवेअर (software) तयार करण्यासाठी विस्तृत डिझाइनचे प्रतिनिधित्व होते. विश्लेषण मॉडेल मॉडेलचा (The analysis model) एक संच आहे जो सिस्टमच्या तांत्रिक प्रतिनिधित्वाशिवाय काहीच नाही. आपण तयार केलेल्या वास्तविक अस्तित्वाची अधिक चांगली समज मिळविण्यासाठी आपण मॉडेल तयार करता. जेव्हा अस्तित्व एक भौतिक गोष्ट आहे, जसे की इमारत, विमान इत्यादी, आपण असे मॉडेल (model) तयार करू शकता जे आकार आणि स्वरूपात एकसारखे आहे परंतु प्रमाणात लहान आहे. जेव्हा ऐंटीटी (entity) सॉफ्टवेअर (software) असते, तेव्हा आपल्या मॉडेलने (model) एक वेगळा फॉर्म घेतला पाहिजे. तांत्रिक स्तरावर, सॉफ्टवेअर अभियांत्रिकी मॉडेलिंग (The analysis modeling) कार्यांच्या मालिकेपासून सुरू होते ज्यामुळे आवश्यकतेचे संपूर्ण तपशील आणि सॉफ्टवेअर (software) तयार करण्यासाठी विस्तृत डिझाइनचे प्रतिनिधित्व होते.

दोन वर्चस्व असलेल्या विश्लेषण मॉडेलिंग तंत्र आहेत:

(The two dominating analysis modeling techniques are)

1. **संरचित विश्लेषण (Structured analysis) :** ही एक शास्त्रीय मॉडेलिंग (modeling) पद्धत आहे. ऑपरेशनल विश्लेषणाचे समाधान करणारे एक संकेत वापरुन, आम्ही मॉडेल तयार करतो जे माहिती (डेटा आणि नियंत्रण) (data and control) सामग्री आणि प्रवाह दर्शवितात, आम्ही सिस्टमला (system) कार्यशील आणि वर्तनात्मकपणे विभाजित करतो आणि आम्ही काय तयार केले पाहिजे त्याचे सार दर्शवितो.
   ऑब्जेक्ट-ओरिएंटेड विश्लेषणाचे उद्दीष्ट म्हणजे ग्राहक-परिभाषित आवश्यकतांचा संच पूर्ण करण्यासाठी संगणक सॉफ्टवेअरचे वर्णन करणाऱ्या मॉडेल्सची मालिका डेव्हलप करणे.
2. **ऑब्जेक्ट -ओरिएंटेड विश्लेषण (Object-oriented analysis) :** संकल्पनांवर आधारित आहे ज्या आपण प्रथम पूर्वी शिकलो - ऑब्जेक्ट आणी अट्रीब्युट, क्लासेस आणि मेंबर, व्होल आणि पार्टस (objects and attributes, classes and members, wholes and parts). या संकल्पना माहिती प्रणालीच्या विश्लेषणावर आणि विशिष्टतेवर लागू करण्यास आम्हाला इतका वेळ का लागला आहे, हा कोणाचाही अंदाज आहे की कदाचित आम्ही पर्यायांचा विचार करण्यासाठी संरचित विश्लेषणाच्या व्यस्त दिवसात प्रवाहानंतर खूप व्यस्त आहोत.

ऑब्जेक्ट -ओरिएंटेड विश्लेषण (Object-oriented analysis) : हेतू म्हणजे सर्व क्लासेस (आणि त्यांच्याशी संबंधित रिलेशनशिप आणी बिहेविअर) (and the relationships and behaviour associated with them) परिभाषित करते, जे समस्येचे निराकरण करण्याच्या समस्येशी संबंधित आहे.

सॉफ्टवेअर अभियांत्रिकीमध्ये (Software Engineering) खाली दिलेल्या दोन प्रकारचे मॉडेल (model) तयार केले आहेत:

(i) **विश्लेषण मॉडेल (Analysis models) :** ते तीन भिन्न डोमेनचे अनुसरण करून सॉफ्टवेअरचे वर्णन करून ग्राहक/वापरकर्त्याच्या आवश्यकतांचे प्रतिनिधित्व करतात:
   a. इनफॉरमेशन डोमेन (Information domain).
   b. फनक्शनल डोमेन, आणि (Functional domain)
   c. बिहेविअरल डोमेन (Behavioral domain)

(ii) **डिझाइन मॉडेल्स (Design models) :** ते सॉफ्टवेअरच्या वैशिष्ट्यांचे प्रतिनिधित्व करतात जे प्रॅक्टिशनर्सना **(practitioners)** प्रभावीपणे बांधण्यास मदत करतात.
   a. द आर्किटेक्चर (The architecture)
   b. द युसर इंटरफेस आणि (The user interface)
   c. कंपोनंन्ट लेवल डिटेल्स (Component-level details)

2.2.3.1 विश्लेषण मॉडेलिंग तत्त्वे (Analysis Modeling Principles) :

सॉफ्टवेअर अभियांत्रिकीमध्ये (software engineering), मोठ्या संख्येने विश्लेषण मॉडेलिंग पद्धती विकसित केल्या आहेत. अन्वेषकांनी (Investigators) विश्लेषण समस्या आणि समस्या कारणे ओळखली आहेत आणि त्यावर मात करण्यासाठी विविध मॉडेलिंग नोटेशन आणि संबंधित सेट हेरिस्टिक्स डेव्हलप केले आहेत.

सर्व विश्लेषण पद्धती ऑपरेशनल तत्त्वांच्या संचाद्वारे संबंधित आहेत. खाली दिलेल्या ॲनालिसिस मॉडेलिंगची ही **ऑपरेशनल तत्त्वे (Principles)** :

1. **समस्येचे माहिती डोमेन प्रतिनिधित्व करणे आणि समजून घेणे आवश्यक आहे (Information domain of a problem must be represented and understood) :** माहिती डोमेनमध्ये, सिस्टममध्ये वाहणारा डेटा, सिस्टममधून वाहणारा डेटा आणि सतत डेटा ऑब्जेक्ट संकलित करणारे आणि आयोजित करणारे डेटा स्टोअर.
2. **सॉफ्टवेअर करत असलेली कार्ये परिभाषित करणे आवश्यक आहे (Functions that the software performs must be defined):** सॉफ्टवेअर फंक्शन्स अंतिम वापरकर्त्यांना थेट लाभ प्रदान करतात. सॉफ्टवेअर फंक्शन्स वापरकर्त्याच्या दृश्यमान असलेल्या वैशिष्ट्यांसाठी अंतर्गत समर्थन देखील प्रदान करते. काही सॉफ्टवेअर फंक्शन्स सिस्टममध्ये वाहणाऱ्या डेटाचे रूपांतर करतात. इतर प्रकरणांमध्ये, सॉफ्टवेअर फंक्शन्स अंतर्गत सॉफ्टवेअर प्रोसेस किंवा बाह्य सिस्टम घटकांवर काही प्रमाणात नियंत्रणावर परिणाम करतात. सॉफ्टवेअर फंक्शन्सचे वर्णन बऱ्याच वेगवेगळ्या स्तरांवर केले जाऊ शकते.
3. **सॉफ्टवेअरच्या वर्तनाचे प्रतिनिधित्व करणे आवश्यक आहे (Behaviour of the software must be represented) :** सॉफ्टवेअरचे वर्तन बाह्य वातावरणासह त्याच्या परस्परसंवादाद्वारे आणि निसर्गाद्वारे चालते. बाह्य प्रणालीद्वारे प्रदान केलेल्या एंड-यूजर्सद्वारे प्रदान केलेला इनपुट डेटा नियंत्रित डेटा, सर्व सॉफ्टवेअर विशिष्ट मार्गाने वर्तन करण्यास कारणीभूत ठरतो.
4. **कार्य, माहिती आणि वर्तन दर्शविणारी मॉडेल्स अशा प्रकारे विभाजने असणे आवश्यक आहे जे स्तरित फॅशनमध्ये तपशील उघडकीस आणते (The models that depict function, information, and behavior must be partitions in such a manner that uncovers details in a layered fashion) :** सॉफ्टवेअर अभियांत्रिकी (software engineering) समस्येचे निराकरण करण्याची पहिली पायरी म्हणजे ऑनालिसिस मॉडेलिंग. ऑनालिसिस मॉडेलिंग प्रॅक्टिशनरला समस्या अधिक चांगल्या प्रकारे समजण्यास आणि समाधानासाठी एक आधार स्थापित करण्यास अनुमती देते. कॉमप्लेक्स आणी क्रीटीकल समस्या सोडवणे कठीण आहे, कारण आम्ही आपण डिव्हाइड आणी कॉनक्यूअर (divide and conquer) ॲप्रोच वापरतो. प्रत्येक उप समस्या

समजण्यास सुलभ होईपर्यंत एक कॉमप्लेक्स आणी क्रीटीकल समस्या उप समस्येमध्ये विभागली जाते. या दृष्टिकोनास पार्टिशनिंग म्हणतात. ऑनालिसिस मॉडेलिंगमध्ये विभाजन करणे ही एक महत्त्वाची स्ट्रॅटेजी आहे.

5. **विश्लेषण कार्य अंमलबजावणीच्या तपशीलांकडे आवश्यक माहितीपासून पुढे जाणे आवश्यक आहे (Analysis task should move from essential information toward implementation detail) :** ऑनालिसिस मॉडेलिंग शेवटच्या युजरच्या दृष्टीकोनातून समस्येचे वर्णन करून सुरुवात होते. निराकरण कसे डेव्हलप केले जाईल किंवा अंमलात आणले जाईल याचा विचार न करता समस्येचे सार स्पष्ट केले आहे. हे समस्येची अंमलबजावणी किंवा विकासाच्या तपशीलांचे सार आहे हे सूचित करते की सार कसे लागू केले जाईल किंवा कसे डेव्हलप केले जाईल. व्हॉईस इनपुटसाठी, व्हीडेओ गेम वापरला जाऊ शकतो.

2.2.3.2 डिझाइन मॉडेलिंग तत्त्वे (Principles) :

- डिझाइन मॉडेल सॉफ्टवेअरसाठी तयार केले गेले आहे, जे सॉफ्टवेअर सिस्टमची विविध भिन्न दृश्ये प्रदान करते. सॉफ्टवेअर डिझाइनचे विविध घटक मिळविण्याच्या पद्धतींची कमतरता नाही. डिझाइन मॉडेलिंगच्या काही पद्धती डेटा- ड्रिव्हन आहेत.
- डेटा चालविलेल्या पद्धती डेटा स्ट्रक्चरला प्रोग्राम आर्किटेक्चर आणि परिणामी प्रोसेस घटकांचे आदेश देण्यास अनुमती देत आहेत. इतर पद्धती पॅटर्न ड्रिव्हन आहेत.
- प्रोसेसिंग पॅटर्न आणि आर्किटेक्चरल स्टाईल डेव्हलप करण्यासाठी प्रॉब्लेम डोमेनबद्दल माहिती वापरुन पॅटर्न चालविलेल्या पद्धती.

डिझाइन मॉडेलिंगचे तत्व खाली सूचीबद्ध आहेत (Principles of design modeling are listed below) :

1. **डिझाइन ऑनालिसिस मॉडेलसाठी शोधण्यायोग्य असावे (The design should be traceable to the analysis model) :** ऑनालिसिस मॉडेल प्रॉब्लेमचे इनफॉरमेशन डोमेन, सिस्टम बिहेविअर, यूजर व्हीजीबल फंक्शन आणि ऑनालिसिस क्लासेसच्या संचाचे वर्णन करते.
   डिझाइन मॉडेल वरील माहितीचे भाषांतर करते:
   (i) आर्किटेक्चर
   (ii) मुख्य कार्ये लागू करणाऱ्या उपप्रणालींचा एक संच आणि
   (iii) घटक स्तराच्या डिझाइनचा एक संच.
2. **डिझाइन नेहमीच सिस्टमच्या आर्किटेक्चरला तयार करण्याचा विचार करते (The design always consider the architecture of the system to be built) :** सॉफ्टवेअर आर्किटेक्चर म्हणजे सॉफ्टवेअर सिस्टमचा सांगाडा किंवा नकाशा आहे. सॉफ्टवेअर आर्किटेक्चर इंटरफेस, प्रोग्राम कंट्रोल फ्लो, डेटा स्ट्रक्चर्स आणि बिहेविअर, ज्या पद्धतीने चाचणी घेता येते, रिझल्टन्ट सिस्टीमची देखभाल करणे इत्यादींवर परिणाम करते. या कारणांसाठी. आर्किटेक्चरल कनसीडरेशनने डिझाइनची सुरुवात झाली पाहिजे.
3. **प्रक्रियेच्या फंक्शन्सच्या डिझाइनइतकी डेटाची रचना तितकीच महत्त्वाची आहे (The design of data is as important as design of processing functions) :** डेटा डिझाइन सॉफ्टवेअर आर्किटेक्चरल डिझाइनचा एक आवश्यक आणि महत्वाचा घटक आहे. एक चांगली स्ट्रक्चर्ड डेटा डिझाइन प्रोग्राम प्रवाह सुलभ करण्यात मदत करते, सॉफ्टवेअर घटकांची रचना आणि अंमलबजावणी सुलभ करते आणि एकूण प्रोसेस अधिक कार्यक्षम आणि प्रभावी करते.
4. **इंटरफेस काळजीपूर्वक डिझाइन केलेले असणे आवश्यक आहे (The interfaces must be designed with care) :** ज्या स्थितीत सिस्टमच्या घटकांमधील डेटा वाहतो त्या त्रुटी प्रसार, डिझाइन साधेपणा आणि प्रोसेस कार्यक्षमतेशी बरेच संबंध आहे. एक चांगले डिझाइन केलेले इंटरफेस इंटिग्रेशन सोपे करते.
5. **यूजर इंटरफेस डिझाइन एंड-यूजरच्या गरजेनुसार ट्यून केले पाहिजे (The user interface design should be tuned to the needs of the end-user) :** यूजर इंटरफेस सॉफ्टवेअरचे दृश्यमान प्रकटीकरण आहे. एक खराब यूजर इंटरफेस डिझाइन बऱ्याचदा सॉफ्टवेअर खराब आहे हे समजते.
6. **कंम्पोनंट लेवल डिझाइन कार्यशीलपणे स्वतंत्र असावे (The component-level design should be functionally independent) :** (फंक्शनल इनडिपेंडन्स) सॉफ्टवेअर घटक किंवा घटकांच्या एकल-विचारसरणीचे

एक उपाय आहे. सॉफ्टवेअर घटकाद्वारे डिलिव्हर्ड केलेली कार्यक्षमता एकत्रित असावी. हे एक आणि फक्त एक फंक्शन किंवा सबफंक्शनवर लक्ष केंद्रित केले पाहिजे.

7. **घटक एकमेकांना आणि बाह्य वातावरणाशी हळूवारपणे जोडले पाहिजेत (The components should be loosely coupled to each other and External environment) :** जागतिक डेटाद्वारे मेसेजिंगद्वारे घटक इंटरफेसद्वारे अनेक प्रकारे जोडणी पूर्ण केली जाते. जेव्हा कपलिंगची पातळी वाढते, तेव्हा त्रुटीचा प्रसार देखील वाढतो आणि सॉफ्टवेअरची एकूण देखभाल कमी होते, या कारणास्तव घटक जोडणी वाजवी आहे तितकी कमी ठेवली पाहिजे.
8. **डिझाइन मॉडेल सहजपणे समजण्यायोग्य असले पाहिजेत (The design models should be easily understandable) :** डिझाइनचा मुख्य हेतू म्हणजे कोड तयार करनाऱ्या प्रॅक्टिशनर्सशी माहिती देणे, जे सॉफ्टवेअरची चाचणी घेतील आणि भविष्यात सॉफ्टवेअर राखू शकतील अशा इतरांना. जर सॉफ्टवेअर मॉडेल डिझाइन समजणे डिफिकल्ट आणि कॉम्प्लेक्स असेल तर ते एक प्रभावी आणि कार्यक्षम संवाद माध्यम म्हणून काम करणार नाही.
9. **डिझाइन पुनरावृत्ती विकसित केली पाहिजे (The design should be developed iteratively) :** सर्व क्रिएटिव्ह ॲक्टिव्हिटीज डिझाइनची पुनरावृत्ती होते. डिझाइनचे प्रथम पुनरावृत्ती डिझाइन आणि रिफाइन डिझाईन एरर्स करेक्ट करण्यासाठी कार्य करतात, परंतु नंतर डिझाइनच्या पुनरावृत्तीने डिझाइन शक्य तितके सोपे करण्यासाठी प्रयत्न केले पाहिजेत.

2.2.4 कन्स्ट्रक्शन (Construction) :

सॉफ्टवेअर अभियांत्रिकीमधील कन्स्ट्रक्शन प्रिन्सिपल ॲक्टीवीटीइजमध्ये कोडिंग आणि टेस्टिंग कार्यांचा एक संच आहे ज्यामुळे ग्राहकांना वितरणासाठी तयार असलेल्या ऑपरेशनल सॉफ्टवेअरला कारणीभूत ठरते.

आपण कोडिंग सुरू करण्यापूर्वी आवश्यक कन्स्ट्रक्शन प्रिन्सिपल खाली सूचीबद्ध आहेत:

आधुनिक सॉफ्टवेअर अभियांत्रिकी कामात, कोडिंगचे अनुसरण केले जाऊ शकते:

1. निराकरण करण्यासाठी प्रॉब्लेम आणी बेसिक डिझाईन तत्वे समजून घ्या.
2. प्रोग्रामिंग भाषा निवडा जी सॉफ्टवेअरच्या गरजा भागवू शकेल आणि ज्या वातावरणात ते कार्य करेल.
3. आपले कार्य सुलभ करू शकणारी साधने निवडा.
4. कंम्पोनंट तयार झाल्यावर युनिट टेस्टिंगचा एक संच तयार करा.

2.2.4.1 कोडिंग प्रिन्सिपल (Coding Principles) :

कोडिंग टास्कला मार्गदर्शन करणारी तत्त्वे प्रोग्रामिंग भाषांसह जवळून संरेखित आहेत. प्रोग्रामिंग शैली आणि प्रोग्रामिंग पद्धती.

अशी अनेक मूलभूत कोडिंग तत्त्वे आहेत जी सांगितले जाऊ शकते

1. प्रीपेरेशन प्रिन्सिपल (Preparation Principles) :

- आपण कोडची एक ओळ लिहिण्यापूर्वी, आपण खात्री करा:
- आपण सोडवण्याचा प्रयत्न करीत असलेल्या समस्येबद्दल समजून घ्या.
- बेसिक डिझाईन तत्त्वे आणि संकल्पना समजून घ्या.
- सॉफ्टवेअर तयार करण्याच्या सॉफ्टवेअरच्या गरजा आणि सॉफ्टवेअर ज्या वातावरणात कार्य करेल त्या वातावरणाची पूर्तता करणारी प्रोग्रामिंग भाषा निवडा.
- प्रोग्रामिंग वातावरण निवडा जे अशी साधने प्रदान करते जी आपले कार्य सोपे आणि सुलभ करेल
- आपण कोड पूर्ण झाल्यावर युनिट चाचण्यांचा एक संच तयार करा जो लागू केला जाईल.

2. कोडिंग प्रिन्सिपल (Coding principles) :

- आपण कोड लिहिण्यास प्रारंभ करताच आपण खात्री करा
- स्ट्रक्चर्ड प्रोग्रामिंग प्रॅक्टिस चा वापर आपल्या अल्गोरिदमला प्रतिबंधित करा
- पेअर प्रोग्रामिंगच्या वापराचा विचार करा.
- डिझाइनच्या गरजा भागविणाऱ्या डेटा स्ट्रक्चर्स निवडा.
- सॉफ्टवेअर आर्किटेक्चर समजून घ्या आणि त्याच्याशी सुसंगत इंटरफेस तयार करा.

- कंडिशनल लॉजिक शक्य तितके सोपे ठेवा.
- अशा प्रकारे नेस्टेड लूप तयार करा ज्यामुळे ते सहजपणे चाचणी करण्यायोग्य बनतील
- अर्थपूर्ण व्हेरिएबल नावे निवडा आणि इतर लोकल कोडिंग स्टँडर्ड मानकांचे अनुसरण करा.
- स्वत: ची दस्तऐवजीकरण (documentation) करणारा कोड लिहा.
- एक व्हिज्युअल लेआउट तयार करा जे समजण्यास मदत करते

3. प्रमाणीकरण तत्त्वे (Validation Principles) :

- आपण आपला पहिला कोडिंग पास पूर्ण केल्यानंतर, आपण सुनिश्चित करा
- योग्य असल्यास कोड वॉकथ्रू आयोजित करा.
- युनिट टेस्टिंग करा आणि आलेले एरर्स करेक्ट करा.
- कोड रीफॅक्टर करा.

2.2.4.2 चाचणी तत्त्वे (Testing Principles) :

ग्लेन मायर्सने असे अनेक नियम नमूद केले आहेत जे टेस्टिंगच्या उद्दीष्टांनुसार चांगल्या प्रकारे सेवा देऊ शकतात:

1. टेस्टिंग ही आलेले एरर्स शोधण्यासाठीची प्रोसेस आहे.
2. एक चांगली टेस्ट केस म्हणजे ज्यात अद्याप न सापडलेले शोधण्याची उच्च संभाव्यता असते.
3. एक यशस्वी टेस्ट केस अशी आहे जी अद्याप न शोधलेले एरर्स उघडकीस आणते.

वरील उद्दीष्टे काही सॉफ्टवेअर डेव्हलपर्सच्या दृष्टिकोनातून नाट्यमय बदल सूचित करतात. ते आता सामान्यत: आयोजित केलेल्या दृश्यास विरोध करतात की यशस्वी चाचणी अशी आहे ज्यामध्ये कोणतीही एरर्स आढळले नाही. जर चाचणी यशस्वीरित्या आयोजित केली गेली तर ते सॉफ्टवेअरमधील एरर्स उघड करेल.

टेस्टिंग असे दर्शविते की सॉफ्टवेअर फंक्शन्स स्पेसिफिकेशननुसार कार्य करत असल्याचे दिसून येते आणि त्या वर्तनात्मक आणि कार्यक्षमतेची आवश्यकता पूर्ण झाल्याचे दिसून येते.

खाली काही टेस्टिंग प्रिन्सिपल आहेत:

1. **सर्व टेस्टिंग ग्राहकांच्या गरजांना शोधण्यायोग्य असाव्यात (All tests should be traceable to customer requirements) : सॉफ्टवेअर टेस्टिंगचे** मुख्य उद्दिष्ट त्रुटी (errors) शोधणे हे आहे, त्यामुळे हे स्पष्ट होते की सर्वात गंभीर दोष (server defects) तेच असतात जे प्रोग्रामला त्याच्या गरजा (requirements) पूर्ण करण्यात अयशस्वी करतात.
2. **टेस्टिंग सुरू होण्यापूर्वीच टेस्टिंग प्लॅन केल्या पाहिजेत (Tests should be planned long before testing begins) :** रिक्वायरमेंट मॉडेल पूर्ण होताच टेस्टिंग प्लॅन सुरू होऊ शकतो. डिझाइन मॉडेल मजबूत झाल्यावर टेस्ट केसची डिटेल्ड डेफिनेशन सुरू होऊ शकते. या कारणास्तव कोणताही कोड जनरेट होण्यापूर्वी सर्व टेस्टिंग प्लॅन आणि डिझाइन केल्या जाऊ शकतात.
3. **पॅरेटो प्रिन्सिपल सॉफ्टवेअर टेस्टिंगवर लागू होते (The Pareto principle applies to software testing) :** या संदर्भात पॅरेटो प्रिन्सिपल सूचित करते की टेस्टिंग दरम्यान उघडकीस आलेल्या सर्व एरर्सपैकी 80% सर्व प्रोग्राम घटकांपैकी 20% ते शोधण्यायोग्य असतील. समस्या अर्थातच या संशयित घटकांना (suspect components) वेगळे करणे आणि त्यांची कसून चाचणी घेणे आहे.
4. **चाचणी "लहान" मध्ये सुरूवात केली पाहिजे आणि "मोठ्या" मध्ये "चाचणीच्या दिशेने प्रगती केली पाहिजे" (Testing should begin "in the small" and progress towards testing "in the large") :** नियोजित आणि एक्झिक्युट झालेल्या पहिल्या चाचण्या सामान्यत: वैयक्तिक घटकांवर (individual components) लक्ष केंद्रित करतात. चाचणी जसजशी प्रगती होते आहे तसतसे घटकांच्या समाकलित क्लस्टर्समध्ये आणि शेवटी संपूर्ण सिस्टममध्ये त्रुटी शोधण्याच्या प्रयत्नात फोकस करते.
5. **संपूर्ण टेस्टिंग करणे शक्य नाही (Exhaustive testing is not possible) :** अगदी मध्यम आकाराच्या प्रोग्रामसाठी पाथ परमिटेशनची संख्या अपवादात्मकपणे मोठी आहे. या कारणास्तव, टेस्टिंग दरम्यान प्रत्येक एक्झिक्युट कॉम्बिनेशन ऑफ पाथ कार्यान्वित करणे अशक्य आहे.

2.2.5 सॉफ्टवेअर डिप्लॉयमेंट (Software Deployment) :

डिप्लॉयमेंट ॲक्टिव्हिटी मध्ये तीन ॲक्टिव्हिटी समाविष्ट असतात म्हणजे डिलिव्हरी सायकल, सपोर्ट सायकल आणि फीडबॅक सायकल.

आजचे सॉफ्टवेअर प्रोसेस मॉडेल्स (software process models) इव्होल्यूशनरी (evolutionary) किंवा इन्क्रिमेंटल (increment) स्वरूपाचे असल्याने, डिप्लॉयमेंट एकदाच होत नाही, तर सॉफ्टवेअर पूर्णत्वाकडे जात असताना अनेक वेळा होते.

1. **डिलिव्हरी सायकल:** प्रत्येक डिलिव्हरी सायकल ग्राहक आणि शेवटच्या वापरकर्त्यांना ऑपरेशनल सॉफ्टवेअर इन्क्रिमेंट (operational software increment) प्रदान करते जे वापरण्यायोग्य फंक्शन्स आणि वैशिष्ट्ये (features) प्रदान करते.
2. **सपोर्ट सायकल:** प्रत्येक सपोर्ट सायकल आजपर्यंतच्या सर्व डिप्लॉयमेंट सायकल दरम्यान सादर केलेल्या सर्व कार्ये आणि वैशिष्ट्यांसाठी डॉक्युमेंटेशन आणि ह्युमन असिस्टंट (human assistance)प्रोव्हाइड करते.
3. **फीडबॅक सायकल:** प्रत्येक फीडबॅक सायकल सॉफ्टवेअर टीमला महत्त्वपूर्ण मार्गदर्शन प्रदान करते ज्याचा परिणाम पुढील वाढीसाठी फंक्शन, वैशिष्ट्ये आणि दृष्टिकोनात बदल होतो.

सॉफ्टवेअर डेवलपमेंट डिप्लॉयमेंट प्रोसेसमध्ये खालील प्रिन्सिपल असतात:

(1) ग्राहकांच्या अपेक्षा किंवा आवश्यकता मॅनेज करा (Manage customer's Expectations or Requirements) : बहुतेक प्रकरणांमध्ये ग्राहकांना त्याच्या/आवश्यकतेनुसार किंवा अपेक्षांपेक्षा पूर्वी सांगितल्यापेक्षा जास्त हवे असते. बऱ्याच प्रकरणांमध्ये ग्राहक त्याच्या सर्व गरजा पूर्ण झाल्यानंतरही निराश होतात. म्हणूनच, सॉफ्टवेअर डिप्लॉयमेंटच्या वेळी, डेवलपर्सकडे ग्राहकांच्या अपेक्षा आणि आवश्यकता (expectations and requirements) व्यवस्थापित करण्याची कौशल्ये असणे आवश्यक आहे

(2) ग्राहकांच्या सपोर्टसाठी रेकॉर्ड-ठेवण्याची यंत्रणा स्थापित करणे आवश्यक आहे (Record-keeping mechanism must be established for customer support) : डिप्लॉयमेंट फेजमध्ये ग्राहक सपोर्ट आवश्यक आणि महत्त्वपूर्ण घटक आहे. सपोर्ट योग्य प्रकारे नियोजित आणि योग्य रेकॉर्ड ठेवण्याच्या यंत्रणेसह केले पाहिजे.

(3) आवश्यक सूचना, डॉक्युमेंटेशन आणि मॅन्युअल प्रदान करा (Provide essential instructions, documentations and manual) : वास्तविक सॉफ्टवेअर प्रोजेक्ट डिलिव्हरीमध्ये सर्व डॉक्युमेंटेशन, हेल्प फाइल्स आणि युजर साठी सॉफ्टवेअर हाताळण्यासाठी मार्गदर्शन समाविष्ट केले पाहिजे.

(4) टेस्ट कम्प्लीट डिलिव्हरी पॅकेज एकत्रित करा (Assemble and test complete delivery package) : कस्टमरच्या आणि डेव्हलपरच्या बाजूने सर्व सहाय्यक आणि आवश्यक मदत मिळणे आवश्यक आहे. या कारणास्तव पुढील सपोर्टसह संपूर्ण एकत्रित आणि टेस्ट केलेल्या डिलिव्हरी पॅकेजसह सीडी डिलिव्हर केली जावी.

i. आवश्यक समर्थित ऑपरेशनल वैशिष्ट्ये आणि मदत.
ii. सॉफ्टवेअर समस्यानिवारणासाठी आवश्यक मॅन्युअल
iii. विकसित सॉफ्टवेअरची सर्व एक्झिक्युटेबल फाइल.
iv. आवश्यक इन्स्टॉलेशन प्रोसेस.

(5) ग्राहकांना कोणतेही सदोष किंवा बग्गी सॉफ्टवेअर डिलिव्हर करू नका (Do not deliver any defective or buggy software to the customer).

वेळेच्या दबावाखाली, काही सॉफ्टवेअर संस्था ग्राहकांना चेतावणी देऊन कमी-गुणवत्तेची इनक्रिमेन्ट प्रदान करतात की बग "पुढील रिलीझमध्ये दुरुस्त केले जातील." "ही एक चूक आहे." सॉफ्टवेअर व्यवसायात एक म्हण आहे: "ग्राहक हे विसरून जातील की तुम्ही उच्च दर्जाचे उत्पादन काही दिवस उशिरा डिलिव्हर केले, परंतु कमी दर्जाच्या उत्पादनामुळे त्यांना झालेल्या समस्या ते कधीही विसरणार नाहीत. सॉफ्टवेअर त्यांना दररोज आठवण करून देते."

2.3 रिक्वायरमेंट इंजिनिअरिंग (Requirement Engineering) :

दर्जेदार सॉफ्टवेअर उत्पादनाची डिझाईन आणि कन्स्ट्रक्शन करण्यासाठी रिक्वायरमेंट इंजिनिअरिंग हा एक पूल आहे. रिक्वायरमेंट इंजिनिअरिंग डेव्हलपर्स, कस्टमर आणि युजर्स मधील संवाद स्थापित करते. रिक्वायरमेंट इंजिनिअरिंग सिस्टमने काय करावे याबद्दल माहिती (रिक्वायरमेंट) गोळा करीत आहे (ते कसे करावे हे नाही).

रिक्वायरमेंट इंजिनिअरिंग मध्ये सॉफ्टवेअरची क्वालिटी सुधारण्यासाठी खालील कार्ये समाविष्ट आहेत.

टास्क 1: **इन्सेप्शन:** प्रॉब्लेम सोडवण्यासाठी समस्येची व्याप्ती आणि स्वरूप परिभाषित करते.

टास्क 2: **एलिसिटेशन:** काय आवश्यक आहे ते परिभाषित करण्यास ग्राहकांना मदत करते.

टास्क 3: **इलॅबोरेशन:** ग्राहकांच्या मूलभूत आवश्यकता परिष्कृत आणि सुधारित केल्या आहेत.

टास्क 4: **नेगोशिएशन:** ग्राहक समस्या परिभाषित करत असताना, प्राधान्यक्रम (priorities) काय आहेत, काय आवश्यक आहे आणि काय नाही, ते कधी हवे आहे, अशा अनेक गोष्टींवर चर्चा करून निगोशिएशन (negotiation) केले जाते.

टास्क 5: **स्पेसिफिकेशन (एसआरएस):** शेवटी, प्रपोज्ड रिक्वायरमेंट स्पेसिफाईड केल्या आहेत. प्रत्येक प्रोजेक्टची रिक्वायरमेंट स्पेसिफिकेशन डॉक्युमेंट आवश्यक आहे कारण हा क्लायंट/एंड युजर्स, बिझनेस ओनर/स्टेकहोल्डर्स आणि प्रोजेक्ट मॅनेजर यांच्यात औपचारिक करार आहे. हे प्रस्तावित प्रकल्पातून अंतिम वापरकर्त्यास काय अपेक्षित आहे ते सांगते.

टास्क 6: **व्हॅलिडेशन:** ग्राहकांचे आणि अभियंता दोघांचेही समस्येचे समजणे सुसंगत असल्याचे सुनिश्चित करते.

टास्क 7: **मॅनेजमेंट:** सॉफ्टवेअर इंजिनियर्सना प्रोजेक्ट जसजसे पुढे जाईल तसतसे कोणत्याही वेळी रिक्वायरमेंटमधील बदलांवर नियंत्रण ठेवण्यास आणि ट्रॅक करण्यास मदत करते.

- सॉफ्टवेअर डेव्हलपमेंट प्रोजेक्ट्समध्ये वरील रिक्वायरमेंट इंजीनियरिंग टास्क महत्त्वपूर्ण आहेत कारण यामुळे डेवलपमेंट कॉस्ट, टाईम, एफर्ट आणि क्वालिटीवर परिणाम होतो.
- एस आर एस मध्ये स्टेकहोल्डर (कस्टमर, मॅनेजर आणि एंड यूजरच्या) मागण्यांचा समावेश आहे. ते प्रस्तावित सॉफ्टवेअरची डिझायर्ड फंक्शन्स, कॉलिटी एट्रिब्यूटस आणि इतर गुणधर्म स्पेसिफाय करतात जे तयार केले जावे किंवा एकत्र केले जावे.
- एसआरएस मुळे सॉफ्टवेअर इंजिनियर्सना ते सोडविण्यासाठी कार्य करणाऱ्या प्रॉब्लेमचे अधिक चांगले समजण्यास मदत करते. एसआरएसमध्ये क्लायंटच्या रिक्वायरमेंटचे विश्लेषण करणे आणि अचूकपणे प्रतिनिधित्व करणे समाविष्ट आहे जे एखाद्या सिस्टममध्ये प्रभावीपणे अंमलात आणले जाऊ शकते जे क्लायंटच्या वैशिष्ट्यांची पुष्टी करेल.
- परंतु, बऱ्याच परिस्थितींमध्ये, योग्य रिक्वायरमेंट स्थापित करण्यात पुरेशी काळजी घेतली जात नाही. यामुळे नंतर, डेव्हलपमेंट लाइफ सायकलमध्ये समस्या उद्भवतात आणि या समस्यांचे निराकरण करण्यासाठी अधिक वेळ आणि पैसा खर्च केला जातो. अशाप्रकारे, आवश्यक आहे की त्यांची अचूकता आणि पूर्णता सुनिश्चित करण्यासाठी पद्धतशीर मार्गाने रिक्वायरमेंट स्थापित केल्या पाहिजेत. रिक्वायरमेंट ॲनालिस्टना कम्युनिकेशन आणि टेक्निकल स्किल दोन्ही आवश्यक आहेत.
- उदाहरणः एक लायब्ररी मॅनेजमेंट सिस्टमची रचना केली जाईल जेणेकरून पुस्तके, सीडी, डीव्हीडी, जर्नल्स इत्यादींची माहिती संग्रहित आणि पुनर्प्राप्त केली जाऊ शकते.

क्लायंटने मागणी केलेल्या रिक्वायरमेंट म्हणजेच 'सिस्टमने काय करावे (Requirements demanded by the client i.e. 'what system should do' are) :

1. **फंक्शनल रिक्वायरमेंट:** शीर्षक, लेखक आणि आयएसडीएन (ISDN) द्वारे शोधणे शक्य आहे.
2. **इम्प्लिमेंटेशन रिक्वायरमेंट:** यूजर इंटरफेस वेब ब्राउझरद्वारे वापरता यावा.
3. **परफॉर्मन्स रिक्वायरमेंट:** प्रति सेकंद किमान 20 ट्रान्सॅक्शन शक्य असावेत.
4. **सिक्युरिटी रिक्वायरमेंट:** युजरने इतर युजरच्या वैयक्तिक डेटामध्ये प्रवेश नसावा.

अशा काही रिक्वायरमेंट आहेत ज्या क्लायंटद्वारे सांगितल्या जात नाहीत परंतु सॉफ्टवेअर इंजिनियरद्वारे निर्णय घेतल्या आहेत:

1. सिस्टीम स्ट्रक्चर.
2. इम्प्लिमेंटेशन टेक्नॉलॉजी आणि लैंग्वेज.
3. डेव्हलपमेंट एन्व्हायरमेंट मेथोडोलॉजी.

डिलिव्हरेबल्स (आउटपुट) ऑफ रिक्वायरमेंट गॅदरिंग टास्क

रिक्वायरमेंट गॅदरिंग टास्क खालील रिझल्ट प्राप्त करण्यात मदत करते:

- आयडेंटिफाइज एनटीटीज
- इसेन्शियल यूज केसेस
- पॉसिबली वर्कफ्लो डायग्राम्स, फ्लो चार्टस
- ई आर मॉडेल - डेटा ऑब्जेक्ट्स (एनटीटीज), ऑब्जेक्ट्सचे प्रॉपर्टीज (एट्रीब्यूटस), ऑपरेशन्स रिलेशनशिप्स बिटवीन ऑब्जेक्ट्स

2.3.1 रिक्वायरमेंट गॅदरिंग ॲनालिसिस (Requirement Gathering and Analysis) :

हे रिक्वायरमेंट, नीड्स आणि कन्स्टंट्रेन्टस कलेक्ट करण्याबद्दल आहे.

हे टास्क कलेक्ट इन्फॉर्मेशन अबाउट:

- डोमेन-प्रोबल्स अँड लॉज (laws)
- एक्झिस्टिंग स्टँडर्डस अँड सिस्टिम्स
- एक्झिस्टिंग स्पेसिफिकेशन्स

प्रश्न आणि उत्तर सत्रामुळे एलिसिटेशन ऑफ रिक्वायरमेंट मिळत नाही आणि म्हणूनच खालील ॲप्रोचेस आवश्यकतेनुसार यशस्वीरित्या काढण्यासाठी (एकत्रित करण्यासाठी) वापरले जातात.

कोलॅबोरेटीव रिक्वायरमेंट्स गॅदरिंग :

- इन्सेप्शन दरम्यान प्रश्न आणि उत्तर सत्र केवळ समस्येची व्याप्ती आणि समाधानाची संपूर्ण धारणा स्थापित करते. या सुरुवातीच्या बैठकीपैकी, स्टेकहोल्डर्स एक किंवा दोन पाने "प्रॉडक्ट रिक्वेस्ट" लिहितात. संमेलनाची जागा, वेळ आणि तारीख निवडली जाते आणि एक सुविधा देणारा निवडला जातो.
- सॉफ्टवेअर टीम आणि इतर स्टेटहोल्डर्सना बैठकीस उपस्थित राहण्यासाठी आमंत्रित केले जाते. बैठकीच्या तारखेपूर्वी प्रॉडक्ट रिक्वेस्ट सर्व उपस्थितांना डिस्ट्रीब्यूट केली जाते.
- सभेला येण्यापूर्वी, प्रत्येक उपस्थितांना सिस्टमच्या सभोवतालच्या वातावरणाचा भाग असलेल्या वस्तूंची यादी तयार करण्यास सांगितले जाते, जे सिस्टमद्वारे तयार केले जावे जे सिस्टमद्वारे त्याचे कार्य करण्यासाठी वापरल्या जातात. तसेच, त्याला या वस्तूंवर कार्य करणाऱ्या सर्विसेस किंवा फंक्शनला सूचीबद्ध करण्यास सांगितले जाते.
- अखेरीस, निर्बंधांची यादी (बिझनेस रुल्स कॉस्ट आणि साईज) आणि परफॉर्मन्स क्रायटेरिया (म्हणजे स्पीड, ॲक्युरेसी) देखील डेव्हलप केले गेले आहेत.
- त्यानंतर वास्तविक बैठक (कोलॅबोरेटीव रिक्वायरमेंट्स गॅदरिंग) तारीख ठरविण्यात येते. या संमेलनात स्टेकहोल्डर्स आणि डेव्हलपर्सची एक टीम प्रॉब्लेम आयडेंटिफाय करण्यासाठी एकत्र काम करते, समाधानाचे घटक प्रस्तावित करते, वेगवेगळ्या पध्दतींशी बोलणी करतात आणि समाधान रिक्वायरमेंट्सचा प्राथमिक संच स्पेसिफाय करतात.

कोलॅबोरेटीव रिक्वायरमेंट्स एकत्रित करणे :

- सॉफ्टवेअर इंजिनीयर आणि कस्टमर दोघेही उपस्थित असलेल्या बैठका आयोजित करणे.
- तयारी आणि सहभागाचे नियम स्थापित केले आहेत.
- सर्व महत्त्वाचे मुद्दे कव्हर करण्यासाठी आणि कल्पनांच्या मुक्त प्रवाहास प्रोत्साहित करण्यासाठी अजेंडा सुचविला जातो.
- एक 'फॅसीलीटेटर (कस्टमर किंवा डेव्हलपर असू शकतो) तो मीटिंगवर नियंत्रण ठेवतो.
- एक 'डेफिनेशन मेकॅनिझम' (वर्कशीट, फ्लिप चार्ट, वॉल स्टिकर्स किंवा इलेक्ट्रॉनिक बुलेटिन बोर्ड, चॅट रूम किंवा व्हर्च्युअल फोरम) असू शकते किंवा वापरली जाते.

द गोल ऑफ सच गॅदरिंग इज :

- प्रॉब्लेम ओळखण्यासाठी
- प्रपोज एलेमेंट्स ऑफ सोल्युशन
- वेगवेगळ्या पध्दतींशी बोलणी करा सोल्यूशन रिक्वायरमेंटचा प्राथमिक संच स्पेसिफाय करा.
- उदाहरणः कन्सिडर "सेफ होम" प्रोजेक्ट.
- या प्रकल्पात गुंतलेल्या वस्तूंची यादी म्हणजे कंट्रोल पॅनेल, स्मोक डिटेक्टर, विंडो आणि डोर सेन्सर, मोशन डिटेक्टर, अलार्म, ॲन इव्हेंट (सेन्सर ॲक्टिव्हेट केला गेला आहे), डिस्प्ले, पीसी, टेलिफोन नंबर, टेलिफोन कॉल इत्यादी.

- सेवांच्या सूचीमध्ये सिस्टम कॉन्फिगर करणे, अलार्म सेट करणे, सेन्सरचे परीक्षण करणे, फोन डायल करणे, नियंत्रण पॅनेल प्रोग्रामिंग करणे आणि प्रदर्शन वाचणे समाविष्ट असू शकते. या सर्व सेवा ऑब्जेक्ट्सवर कार्य करतात.
- सेन्सर ऑपरेट करत नसताना सिस्टमने ओळखणे आवश्यक आहे म्हणून लिस्ट ऑफ कनस्ट्रेन्टस असू शकते, यूजर फ्रेंडली असणे आवश्यक आहे, स्टँडर्ड फोन लाइनवर डायरेक्ट इंटरफेस करणे आवश्यक आहे.
- सेन्सर इव्हेंट असताना परफॉर्मन्स क्रायटेरिया दोन सेकंदाच्या आत असू शकतो, इव्हेंट प्रायोरिटी स्कीम अंमलात आणली पाहिजे.
- जसजसे मेळावे सुरू होते तसतसे चर्चेचा पहिला विषय म्हणजे नवीन उत्पादनाची गरज आणि जस्टीफिकेशन - प्रत्येकाने हे मान्य केले पाहिजे की प्रॉडक्ट जस्टीफाईड आहे. एकदा ॲग्रीमेंट एस्टॅब्लिश झाल्यानंतर, प्रत्येक सहभागी आपली यादी चर्चेसाठी सादर करतो. याद्या खोलीच्या भिंतींवर पिन केल्या जाऊ शकतात आणि त्या इलेक्ट्रॉनिक बुलेटिन बोर्डवर किंवा चॅट रूमच्या वातावरणात देखील पोस्ट केल्या जाऊ शकतात. प्रत्येक लिस्टेड एन्ट्री स्वतंत्रपणे हाताळण्यास सक्षम असावी (डिलीटेड, ॲडेड किंवा मॉडिफाइड). या टप्प्यावर, टीका आणि वादविवाद (debate) यांवर सक्त मनाई आहे. त्यानंतर, समूहाद्वारे एकत्रित यादी तयार केली गेली आहे जेणेकरून निरर्थक नोंदी दूर केल्या जातील आणि चर्चेदरम्यान येणाऱ्या कोणत्याही नवीन कल्पना जोडल्या पाहिजेत. पुढे, फॅसिलिटेटर चर्चेचे समन्वय करतो, एकत्रित यादी (combined list) लहान करण्यासाठी, लांबवण्यासाठी किंवा पुन्हा शब्दबद्ध (reword) करण्यासाठी जेणेकरून डेव्हलप केल्या जाणाऱ्या सिस्टमचे (system) योग्यरित्या प्रतिबिंब दिसेल.
- आता, संघ लहान उप -संघांमध्ये विभागला गेला आहे; प्रत्येक लिस्टच्या एक किंवा अधिक एंट्रीसाठी मिनी स्पेसिफिकेशन डेव्हलप करण्यासाठी प्रत्येक कार्य करते. प्रत्येक मिनी तपशील सूचीमध्ये समाविष्ट केलेल्या शब्दाचे विस्तार आहे.
- उदाहरणार्थ, कंट्रोल पॅनेलचे मिनी स्पेसिफिकेशन - हे एक वॉल माऊंटेड युनिट आहे आणि ते 9*5 इंच आकाराचे आहे, सेन्सर आणि पीसीशी वायरलेस कनेक्टिव्हिटी आहे, युजर इंटरॅक्शन 12 की असलेल्या कीबोर्डद्वारे होतो, 2*2 इंच सीडी डिस्प्ले यूजर फीडबॅक प्रोव्हाइड करतो, इंटरॅक्टिव्ह प्रॉम्प्टस, इको अँड सिमिलर फंक्शन्स प्रोव्हाइड करतो.
- मिनी-स्पेक पूर्ण झाल्यानंतर, प्रत्येक पार्टीसीपंट सिस्टमसाठी व्हॅलिडेशन क्रायटेरियाची यादी बनवितो. या सर्व याद्या एकत्रित केल्या आहेत आणि व्हॅलिडेशन क्रायटेरियाची कम्बाईनड यादी तयार केली आहे. अखेरीस, एक किंवा अधिक पार्टीसीपंटना सभेच्या सर्व इनपुटचा वापर करून संपूर्ण ड्राफ्ट स्पेसिफिकेशन लिहिण्याचे कार्य नियुक्त केले जाते.

2.3.1.1 क्वालिटी फंक्शन डिप्लॉयमेंट (Quality Function Deployment) :

- ॲक्युरेट रिक्वायरमेंट गॅदरिंग करण्यात मदत करणारे एखादे टेक्निक असणे खूप उपयुक्त ठरेल. क्वालिटी फंक्शन डिप्लॉयमेंट (क्यू एफ डी) हे एक टेक्निक आहे जे सॉफ्टवेअर टीमला कस्टमर वॉन्टस आणि नीड्स स्पष्टपणे स्पेसिफाय करण्यास इनेबल करते आणि नंतर त्या गरजा पूर्ण करण्याच्या परिणामाच्या प्रत्येक प्रपोज्ड ऑब्जेक्ट किंवा सर्विस क्षमतेचे मूल्यांकन करते.
- क्यूएफडी हे एक टेक्निक आहे जे सॉफ्टवेअरसाठी कस्टमर रिक्वायरमेंट टेक्निकल रिक्वायरमेंटमध्ये भाषांतर करते.
- क्यूएफडी संपूर्ण सॉफ्टवेअर इंजिनिअरिंग प्रोसेसमध्ये कस्टमरच्या गरजेवर लक्ष केंद्रित करते.

क्यू एफ डी 3 प्रकारच्या रिक्वायरमेंट ओळखते :

1. नॉर्मल रिक्वायरमेंट
2. एक्सपेक्टेड रिक्वायरमेंट
3. एक्सायटिंग रिक्वायरमेंट

1. नॉर्मल रिक्वायरमेंट (Normal requirements):

हे कस्टमरच्या बैठकीत सिस्टमसाठी नमूद केलेली ऑब्जेक्टिव्ह आणि गोल्स (objectives and goals) रिफ्लेक्ट करते. जर या रिक्वायरमेंट उपस्थित असतील तर ग्राहक समाधानी आहे.

उदाहरणः ग्राफिकल डिस्प्ले, विशिष्ट सिस्टम फंक्शन्स आणि परिभाषित लेव्हल्स ऑफ परफॉर्मन्स (defined levels of performance).

2. एक्सपेक्टेड रिक्वायरमेंट (Expected requirements):

हे सिस्टीमसाठी इम्प्लिसिट (implicit) असतात आणि इतके मूलभूत असू शकतात की ग्राहक ते स्पष्टपणे नमूद करत नाहीत. अशा रिक्वायरमेन्ट्स (requirements) नसल्यास ग्राहकांमध्ये असमाधान निर्माण होते.

उदाहरणार्थ: ह्युमन-मशीन इंटरॅक्शनची (human-machine interaction) सुलभता, एकूणच ऑपरेशनल करेक्टनेस (correctness) आणि रिलायबिलिटी (reliability) आणि सॉफ्टवेअर इन्स्टॉलेशनची (software installation) सुलभता.

3. एक्सायटिंग रिक्वायरमेंट (Exciting requirements):

हे असे फिचर्स (features) दर्शवतात जे ग्राहकांच्या अपेक्षांच्या पलीकडचे असतात आणि जेव्हा ते उपस्थित असतात तेव्हा खूप समाधानकारक ठरतात.

उदाहरणार्थ: ग्राहकाला वर्ड प्रोसेसिंग सॉफ्टवेअर (Word processing software) मानक फिचर्ससह (features) हवे असेल, परंतु जेव्हा त्याला मिळालेल्या प्रोडक्टमध्ये (product) मूलभूत फिचर्ससह अनेक पेज लेआउट कॅपॅबिलिटीज (page layout capabilities) दिसतात, तेव्हा तो खूश होतो. ग्राहकांसोबतच्या मीटिंगमध्ये, फंक्शन डिप्लॉयमेंट (function deployment) सिस्टीमसाठी (system) आवश्यक असलेल्या प्रत्येक फंक्शनचे (function) मूल्य निर्धारित करते. इन्फॉर्मेशन डिप्लॉयमेंट (information deployment) डेटा ऑब्जेक्ट्स (data objects) आणि इव्हेंट्स (events) ओळखते जे सिस्टीमने वापरले पाहिजेत आणि तयार केले पाहिजेत. शेवटी, टास्क डिप्लॉयमेंट (task deployment) सिस्टीमच्या (system) वर्तणुकीची तपासणी करते. त्यानंतर, तीनही डिप्लॉयमेंट्सदरम्यान (deployments) निर्धारित केलेल्या रिक्वायरमेन्ट्सच्या (requirements) सापेक्ष प्राधान्य (priority) निश्चित करण्यासाठी व्हॅल्यू ॲनालिसिस (value analysis) केले जाते. क्यूएफडी (QFD) कस्टमर इंटरव्ह्यूज (customer interviews) आणि ऑब्झर्व्हेशन (observation), सर्वेक्षण आणि ऐतिहासिक डेटाची (data) तपासणी रिक्वायरमेन्ट्स गॅदरिंग ॲक्टिव्हिटीसाठी (requirements gathering activity) रॉ डेटा (raw data) म्हणून वापरते. त्यानंतर हा डेटा रिक्वायरमेन्ट्सच्या (requirements) सारणीमध्ये (table) रूपांतरित केला जातो ज्याला कस्टमर व्हॉईस टेबल (customer voice table) म्हणतात. अपेक्षित रिक्वायरमेन्ट्स काढण्यासाठी आणि रोमांचक रिक्वायरमेन्ट्स मिळवण्याचा प्रयत्न करण्यासाठी विविध डायग्राम्स (diagrams), मॅट्रिक्स (matrices) आणि इव्हॅल्युएशन मेथड्सचा (evaluation methods) वापर केला जातो.

क्यू एफ डी चे फायदे : (Benefits of QFD)

- युजरचा सहभाग सुधारतो.
- मॅनेजमेंट सपोर्ट आणि इन्व्हॉलमेंट सुधारते.
- डेव्हलपमेंट ची लाइफ सायकल लहान करते.
- प्रोजेक्ट डेव्हलपमेंट सुधारतो.
- टीम इनव्हॉल्वमेंटला समर्थन देते.
- स्ट्रक्चर्स कम्युनिकेशन प्रोसेस.
- गुणवत्ता सुधारण्यासाठी एक प्रतिबंधात्मक साधन प्रदान करते.
- इन्फॉर्मेशन लॉस टाळते

2.3.1.2 डेव्हलपिंग युज केस सीन्यारीओज (Developing Use Case Scenarios) :

सराव मध्ये एक सिस्टीम कशी वापरली जाते याचे वर्णन सीन्यारीओज आहे. सीन्यारीओज ही सिस्टीम कशी वापरली जाऊ शकते याची वास्तविक जीवनाची उदाहरणे आहेत. आउटलाइन रिक्वायरमेंट डिस्क्रिप्शन जोडण्यासाठी सीन्यारीओज विशेषतः उपयुक्त आहेत. सीन्यारीओज आउटलाइन ऑफ इंटरॅक्शनने सुरू होते आणि एलिसिटेशन दरम्यान, त्या इंटरॅक्शनचे संपूर्ण वर्णन तयार करण्यासाठी तपशील जोडला जातो. सर्वसाधारणपणे, सीन्यारीओज मध्ये हे समाविष्ट आहे:

- सीन्यारीओजच्या सुरूवातीस सिस्टम स्टेट आणि वापरकर्त्याच्या अपेक्षांचे वर्णन.
- घटनांचा सामान्य प्रवाह.
- काय चूक होऊ शकते आणि हे कसे हाताळले जाऊ शकते.
- इतर कनकरंट ॲक्टिव्हिटीजबद्दल माहिती.

- सीन्यारीओ पूर्ण झाल्यावर सिस्टम स्टेटचे वर्णन.
- डिफरंट युजरचा संग्रह सीन्यारीओजच्या युज केस बनवतात **.**
- प्रत्येक सीन्यारीओचे वर्णन ॲक्टर परसनच्या दृष्टिकोनातून किंवा सॉफ्टवेअरशी थेट किंवा अप्रत्यक्षपणे संवाद साधणाऱ्या डिव्हाइसवरून केले जाते.

प्रत्येक सीन्यारीओ खालील प्रश्नांची उत्तरे देते (Each scenario answers the following questions) :

- प्रायमरी आणि सेकंडरी ॲक्टर कोण आहेत?
- ॲक्टरचे ध्येय काय आहेत?
- यूजर स्टोरी सुरू होण्यापूर्वी कोणत्या प्रीकंडिशन्स अस्तित्वात असाव्या?
- ॲक्टरने कोणती मुख्य कार्ये केली आहेत?
- कोणत्या अपवादांचा विचार केला जाऊ शकतो?
- ॲक्टरच्या परस्परसंवादामध्ये कोणते बदल शक्य आहेत?
- ॲक्टरला सिस्टीमकडून कोणती माहिती पाहिजे आहे?
- बाह्य वातावरणातील बदलांविषयी ॲक्टरला सिस्टमला माहिती द्यावी लागेल का?
- ॲक्टरला अनपेक्षित बदलांविषयी माहिती द्यावी अशी इच्छा आहे का?

उदाहरणः सेफ होम प्रोजेक्ट

या उदाहरणात, आम्ही घरमालकांना प्रायमरी ॲक्टर मानतो; आणि सिस्टम ॲडमिनिस्ट्रेटर आणि सेन्सर हे सेकंडरी ॲक्टर आहेत.

घरमालक नियंत्रण पॅनेलचा वापर करून सेफ होम सिक्युरिटी फंक्शनशी संवाद साधतो.

सेफ होम सिक्युरिटी सिस्टम ॲक्टिवेशनसाठी मूलभूत यूज-केस टेम्पलेट खालीलप्रमाणे आहे:

यूज-केस	**सिक्युरिटी मॉनिटरिंग सिस्टीम आरंभ करा**
प्रायमरी ॲक्टर	घरमालक
गोल इन कॉनटेक्स्ट	जेव्हा घरमालक घर सोडतो तेव्हा सिक्युरिटी मॉनिटरिंग सिस्टीम सेन्सर सेट करण्यासाठी.
प्रिकंडिशन्स	सिस्टमचा पासवर्ड सेट करणे आणि विविध सेन्सर ओळखणे आवश्यक आहे. अलार्म फंक्शन्सचालू करण्यासाठी घरमालक डीकोड करते.
ट्रिगर	अलार्म फंक्शन्स चालू करण्यासाठी घरमालक डीकोड करते.
सीन्यारीओ	घरमालक कंट्रोल पॅनेलचे निरीक्षण करते. घरमालक पासवर्ड एंटर करतो. घरमालक 'मुक्काम' किंवा 'दूर निवडतो**.** घरमालक आपत्कालीन परिस्थितीत पॅनीक बटण दाबतो. घरमालक सिक्युरिटी सिस्टीम सक्रिय करते.
एक्सेप्शन्स	कोणतेही सेन्सर खुले असल्यास नियंत्रण पॅनेल तयार नाही. पासवर्ड चुकीचा आहे. मुक्काम निवडला आहे आणि परिमीटरसेन्सर सक्रिय आहेत. दूर निवडले गेले आहे आणि सर्व सेन्सर सक्रिय आहेत.
इंटरफेस	कंट्रोल पॅनल
सेकंडरी ॲक्टर	सपोर्ट टेक्निशियन, सेन्सर.
केरीज/ इशूज	संकेतशब्द वापरल्याशिवाय आम्ही सिस्टम सक्रिय करू शकतो**?**

यूज-केस	सिक्युरिटी मॉनिटरिंग सिस्टीम आरंभ करा
	सिस्टम प्रत्यक्षात सक्रिय होण्यापूर्वी निष्क्रिय करण्याचा कोणताही मार्ग आहे का?
इंटरफेस	नियंत्रण पॅनेल.
सेकंडरी ॲक्टर	सपोर्ट टेक्निशियन, सेन्सर.

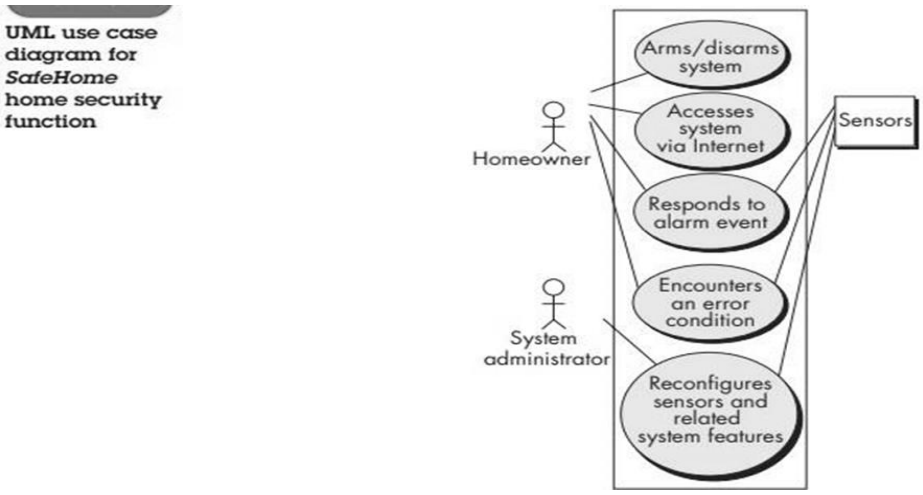


Fig. 2.1: द युज केस डायग्राम फॉर सेफ होम सिक्युरिटी फंक्शन (The Use Case Diagram for Safe Home Security Function)

2.3.1.3 एलिसिटेशन वर्क प्रॉडक्ट्स (Elicitation Work Products) :

हे रिक्वायरमेंटसचा रिजल्ट म्हणून डेव्हलप केले गेले आहे आणि ते तयार करण्याच्या सिस्टमच्या आकारानुसार ते बदलते. हे वर्णन करते:

- सिस्टमची गरज आणि व्यवहार्यता (The need and feasibility of the system).
- सिस्टमची व्याप्ती (The scope of the system).
- कस्टमर, युजर आणि इतर स्टेकहोल्डर्सची यादी ज्यांनी रिक्वायरमेंटसनुसार भाग घेतला.
- आवश्यक असलेले तांत्रिक वातावरण (The technical environment i.e. required).
- रिक्वायरमेंट्सची यादी (त्यांच्या कार्यपद्धतीनुसार आयोजित) आणि प्रत्येक रिक्वायरमेंट्सना लागू होणाऱ्या डोमेन मर्यादा (domain constraints).
- वेगवेगळ्या ऑपरेटिंग कंडिशन्समध्ये सिस्टमच्या वापराविषयी माहिती प्रदान करणाऱ्या युजरच्या सीन्यारीओचा एक संच.
- रिक्वायरमेंटस अधिक चांगल्या प्रकारे डिफाइन करण्यासाठी डेव्हलप केलेले कोणतेही प्रोटोटाइप.
- यापैकी प्रत्येक वर्क प्रॉडक्टचे रिक्वायरमेंट्स एलिसिटेशनमध्ये सहभागी असलेल्या सर्व लोकांकडून रिव्ह्यू (reviewed) केले जाते.

2.3.2 रिक्वायरमेंट्सचे प्रकार (Types of Requirements):

सॉफ्टवेअर सिस्टम रिक्वायरमेंटस बऱ्याचदा फंक्शनल रिक्वायरमेंट आणि नॉन फंक्शनल रिक्वायरमेंट म्हणून वर्गीकृत केल्या जातात.

फंक्शनल रिक्वायरमेंट सिस्टमने प्रदान केलेल्या सेवांचे स्टेटमेन्ट्स आहेत, विशिष्ट इनपुटवर सिस्टमने काय प्रतिक्रिया द्यावी आणि विशिष्ट परिस्थितीत सिस्टमने कसे वागावे.

नॉन फंक्शनल रिक्वायरमेंट सिस्टमद्वारे ऑफर केलेल्या सर्विसेस किंवा फंक्शन्स यावर मर्यादा आहेत. त्यामध्ये वेळेची मर्यादा, विकास प्रक्रियेवरील अडचणी आणि मानकांचा समावेश आहे. नॉन-फंक्शनल रिक्वायरमेंट संपूर्णपणे सिस्टमवर लागू होते.

Fig. 2.2: सॉफ्टवेअर इंजीनियरिंगमध्ये विविध प्रकारच्या रिक्वायरमेंट दर्शविते. सिस्टमसाठी फंक्शनल रिक्वायरमेंट सिस्टमने काय करावे हे वर्णन करते. नावानुसार नॉन-फंक्शनल रिक्वायरमेंट ही रिक्वायरमेंट आहेत जी सिस्टमद्वारे यूजर्सकडे डिलिव्हर्ड केलेल्या विशिष्ट सेवांशी थेट संबंधित नाहीत.

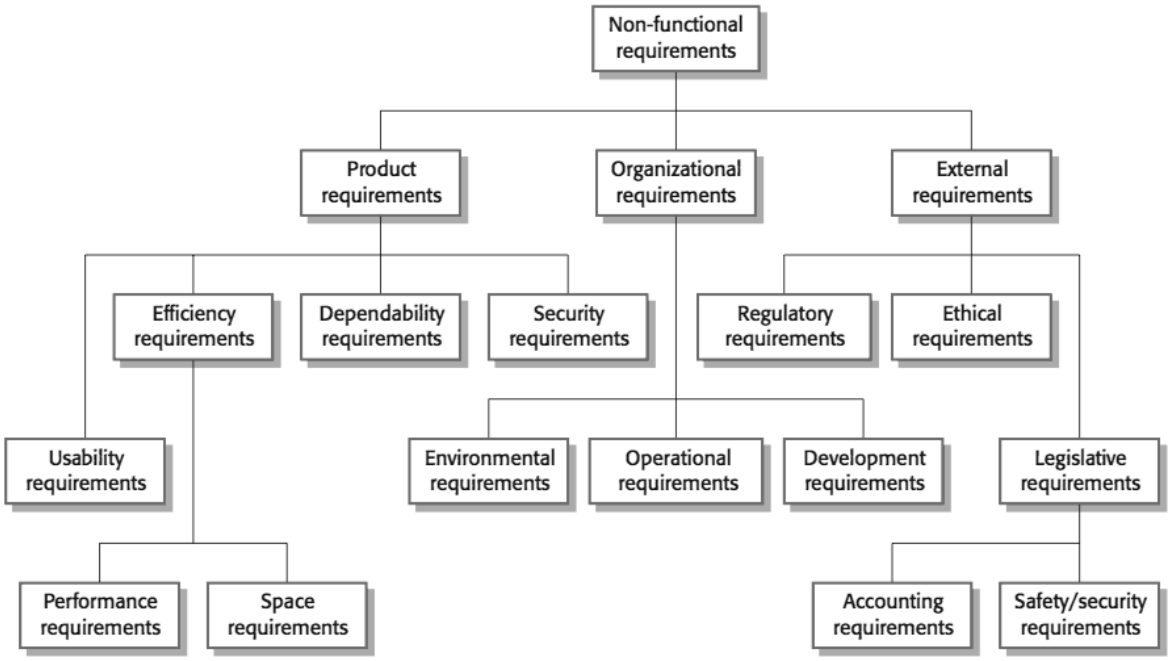


Fig. 2.2: नॉन फंक्शनल रिक्वायरमेंट (Non-Functional Requirement)

2.3.2.1 फंक्शनल रिक्वायरमेंट (Functional Requirements):

आयईईई (IEEE) फंक्शनल रिक्वायरमेंट म्हणून परिभाषित करते, "एक फंक्शन जे सिस्टम किंवा कंपोनंट करण्यास सक्षम असेल". फंक्शनल रिक्वायरमेंट त्याच्या वातावरणासह सॉफ्टवेअरच्या परस्परसंवादाचे वर्णन करतात आणि सॉफ्टवेअरमध्ये समाविष्ट केलेली इनपुट, आउटपुट, एक्स्टर्नल इंटरफेस आणि फंक्शन्स समाविष्ट असतात. फंक्शनल रिक्वायरमेंट्समध्ये (functional requirements) प्रदान केलेल्या सेवांचाही समावेश होतो, ज्या विशिष्ट इनपुट्स (inputs) किंवा विशिष्ट परिस्थितीत सॉफ्टवेअरने कसे प्रतिक्रिया द्यावी हे स्पष्ट करतात. या रिक्वायरमेंट पूर्ण आणि सुसंगत असाव्यात. कम्प्लीटनेस असे सूचित करते की सर्व यूजरच्या रिक्वायरमेंट परिभाषित केल्या आहेत. कन्सिस्टन्सी सूचित करते की कोणत्याही कॉन्ट्राडिक्टरी व्याख्येशिवाय सर्व रिक्वायरमेंट स्पष्टपणे स्पेसिफाय केल्या आहेत.

2. नॉन-फंक्शनल रिक्वायरमेंट (Non-Functional Requirements):

रिलायबिलिटी, युसिबिलिटी, पोर्टेबिलिटी इफिसिएनसी, परफॉर्मन्स, रिस्पॉन्स टाईम इत्यादीसारख्या सिस्टम ॲट्रीब्यूटसशी संबंधित क्वालिटी रिक्वायरमेंट म्हणून ओळखल्या जाणाऱ्या नॉन-फंक्शनल रिक्वायरमेंट. या रिक्वायरमेंटस यूजरच्या रिक्वायरमेंटस, बजेट कन्स्ट्रेन्ट्स, ऑर्गनायझेशनल पॉलिसीज इत्यादींमुळे उद्भवतात. या रिक्वायरमेंटस सिस्टमद्वारे प्रोव्हाइड केलेल्या कोणत्याही विशिष्ट कार्याशी थेट संबंधित नाहीत. वेगवेगळ्या प्रकारच्या नॉन-फंक्शनल आवश्यकता खाली स्पष्ट केल्या आहेत:

2.3.2.2 प्रॉडक्ट रिक्वायरमेंटस (Product Requirements):

प्रॉडक्ट रिक्वायरमेंटस सॉफ्टवेअरचे बिहेवियर स्पेसिफाय करते. या रिक्वायरमेंटस सॉफ्टवेअर प्रॉडक्ट कसे कार्य करते हे स्पेसिफाय करते.

प्रॉडक्ट रिक्वायरमेंटसमध्ये पुढील गोष्टींचा समावेश आहे:

1. रिलायबिलिटी रिक्वायरमेंटस सॉफ्टवेअरच्या एक्सेप्टेबल फेल्लूअर रेटचे (acceptable failure rate) वर्णन करतात. उदाहरणार्थ, एखादा धोका उद्भवला तरीही सॉफ्टवेअर ऑपरेट करण्यास सक्षम असावे.
2. युसिबिलिटी रिक्वायरमेंटस युजर्स सॉफ्टवेअर ऑपरेट करण्यास सक्षम असलेल्या सहजतेचे वर्णन करतात. उदाहरणार्थ, सॉफ्टवेअर कमी कीस्ट्रोक आणि माउस क्लिकसह कार्यक्षमतेमध्ये प्रवेश प्रदान करण्यास सक्षम असावे.

3. इफिसिएनसी रिक्वायरमेंटस सॉफ्टवेअर संसाधनांचा इष्टतम वापर किती प्रमाणात करते, सिस्टम ज्या स्पीडसह कार्यान्वित करते आणि त्याच्या ऑपरेशनसाठी ती वापरणारी मेमरी यांचे वर्णन करते. उदाहरणार्थ, सिस्टम एक्झिस्टिंग सिस्टीम (existing system) पेक्षा कमीतकमी तीन पट वेगवान ऑपरेट करण्यास सक्षम असावे.
4. पोर्टेबिलिटी रिक्वायरमेंटस सॉफ्टवेअर एका प्लॅटफॉर्मवरुन दुसऱ्या प्लॅटफॉर्मवर ट्रान्सफर केल्या जाऊ शकतात अशा सहजतेचे वर्णन करतात. उदाहरणार्थ, संपूर्ण सॉफ्टवेअरचे पुन्हा डिझाइन न करता सॉफ्टवेअरला भिन्न ऑपरेटिंग सिस्टममध्ये पोर्ट करणे सोपे आणि साधे असले पाहिजे.

2.3.2.3 ऑर्गनायझेशनल रिक्वायरमेंटस (Organizational Requirements):

ऑर्गनायझेशनल रिक्वायरमेंटस एखाद्या ऑर्गनायझेशनच्या पॉलिसीज आणि प्रोसेजरमधून प्राप्त केल्या जातात.
ऑर्गनायझेशनल रिक्वायरमेंटस खालीलप्रमाणे आहेत:

1. इम्प्लिमेंटेशन रिक्वायरमेंटस प्रोग्रामिंग भाषा आणि डिझाइन पद्धत यासारख्या रिक्वायरमेंटसचे वर्णन करते.
2. स्टँडर्ड रिक्वायरमेंटस डिस्क्राइब करतात सॉफ्टवेअर डेव्हलपमेंट दरम्यान वापरल्या जाणाऱ्या प्रोसेसचे स्टँडर्ड. उदाहरणार्थ, आयएसओ (इंटरनॅशनल ऑर्गनायझेशन फॉर स्टँडर्डडिझेशन) आणि आयईईई (IEEE) स्टँडर्डद्वारे स्पेसिफाय केलेल्या स्टँडर्डचा वापर करून हे सॉफ्टवेअर डेव्हलप केले जावे.
3. डिलिव्हरी रिक्वायरमेंटस सॉफ्टवेअर आणि त्याचे डॉक्युमेंटेशन युजरला केव्हा डिलिव्हर केले जाईल हे स्पेसिफाय करते.

2.3.2.4 एक्स्टर्नल रिक्वायरमेंटस (External Requirements):

एक्स्टर्नल रिक्वायरमेंटस मध्ये सॉफ्टवेअर किंवा त्याच्या डेव्हलपमेंट प्रोसेसवर एक्स्टर्नली प्रभावित करणाऱ्या सर्व रिक्वायरमेंटस समाविष्ट आहेत.
एक्स्टर्नल रिक्वायरमेंटस खालीलप्रमाणे आहेत:

1. इंटरऑपरेबिलिटी रिक्वायरमेंटस एक किंवा अधिक ऑर्गनायझेशनमध्ये भिन्न संगणक-आधारित सिस्टीम एकमेकांशी संवाद साधतात त्या मार्गाची व्याख्या करते.
2. लेजिस्लेटिव्ह रिक्वायरमेंटस कार्यक्षेत्रात सॉफ्टवेअर लीगल जुरीसडिक्शनने चालविते. उदाहरणार्थ, पायरेटेड सॉफ्टवेअर विकले जाऊ नये.
3. इथिकल रिक्वायरमेंटस सॉफ्टवेअरचे रुल्स आणि रेगुलेशन्स स्पेसिफाइस करते जेणेकरून ते युजरसाठी ॲक्सेप्टेबल असतील.

2.3.2.5 एलिसिटिंग रिक्वायरमेंटस (Eliciting Requirements):

सॉफ्टवेअर, सिस्टम आणि व्यवसायाबद्दल माहिती शोधण्याबद्दल एलिसिटेशन आहे. ग्राहकांना आणि वापरकर्त्यांशी त्यांची रिक्वायरमेंटस काय आहे हे निर्धारित करण्यासाठी संवाद साधण्याचे कार्य आवश्यक आहे.
खालील माहिती ग्राहक आणि वापरकर्त्यांकडून शोधली जाऊ शकते:

1. दररोज सॉफ्टवेअर कसे वापरावे**?**
2. सॉफ्टवेअरद्वारे काय साध्य करावे**?**
3. सॉफ्टवेअर उत्पादनाची ऑब्जेक्टिव्ह आणि गोल्स कोणती आहेत**?**
4. सॉफ्टवेअर व्यवसायाच्या गरजेनुसार कसे बसते**?**

वापरकर्त्याच्या आवश्यकतांबद्दल माहिती काढण्यासाठी वापरल्या जाणाऱ्या आवश्यकतेचे प्रश्नः

1. Who?	ऑर्गनायझेशनल युनिट, नोकरीचा प्रकार किंवा क्लायंटबद्दल जाणून घेणे ज्याच्याशी रिक्वायरमेंटस संबंधित आहे
2. What?	काम करण्यासाठी आवश्यक असलेल्या कार्यांविषयी जाणून घेणे म्हणजेच फंक्शनल रिक्वायरमेंटस
3. Where?	मशीनच्या प्रकाराबद्दल (उदा. क्लायंट, सर्व्हर इ.) जाणून घेण्यासाठी ज्यावर प्रोसेस करणे आवश्यक आहे.
4. When?	या प्रश्नाद्वारे एखाद्याला रिक्वायरमेंटस अनेक पैलू माहित असू शकतात. काही उदाहरणे आहेत

	परफॉर्मन्स: कार्य सुरू झाल्यानंतर किती वेळात पूर्ण केले पाहिजे ? ऑर्डर: कोणत्या भिन्न अनुक्रमात कार्ये सुरू करणे आणि पूर्ण करणे आवश्यक आहे ? कालावधी: प्रक्रियेच्या डेटाचा कालावधी किती कालावधी आहे ज्यावर प्रोसेसिंग लॉजिक लागू केले जावे ?
5. Why?	सॉफ्टवेअरची गरज का आहे या रिक्वायरमेंटस मागील तर्क जाणून घेण्यासाठी.
6. How?	कार्य कसे सुरू केले जाते आणि कसे परफॉर्म केले जाते आणि परिणाम कसे डिलिव्हर केले जातात हे जाणून घेण्यासाठी. "कसे" विचारणे फंक्शनालिटी डिलिव्हर करण्यासाठी वापरल्या जाणाऱ्या पद्धती निर्धारित करते. हे डेटा एंट्री, डेटा प्रोसेसिंग आणि डीसीमिनेशन ऑफ इन्फॉर्मेशनसाठी यंत्रणा समजून घेण्याशी संबंधित आहे.

प्रोसेस ऑफ रिक्वायरमेंट एलिसिटेशन (Process of Requirement Elicitation) :

1. रिक्वायरमेंट गॅदरिंग
2. ऑर्गनायझेशन रिक्वायरमेंट
3. नेगोसिएशन अँड डिस्कशन
4. रिक्वायरमेंट स्पेसिफिकेशन

1. **रिक्वायरमेंट गॅदरिंग:** डेव्हलपर्स क्लायंट आणि शेवटच्या वापरकर्त्यांशी चर्चा करतात आणि सॉफ्टवेअरकडून त्यांच्या अपेक्षा जाणून घेतात.
2. **रिक्वेस्टमेंट ऑर्गनायझेशन:** डेव्हलपर्स इम्पॉर्टन्स,अर्जुनसी आणि कन्व्हिनिअन्ससाठी आवश्यकतेनुसार रिक्वायरमेंटला प्राधान्य देतात आणि त्यांची व्यवस्था करतात.
3. **नेगोशिएशन अँड डिस्कशन:** जर रिक्वायरमेंट ॲम्बीगुएस असतील किंवा विविध स्टेकहोल्डर्सच्या आवश्यकतांमध्ये (requirements) काही संघर्ष असतील तर ते असल्यास, त्यास निगोशिएट केली जाते आणि स्टेकहोल्डर्सच्याशी चर्चा केली जाते. नंतर आवश्यकतांना (requirements) प्राधान्य दिले जाऊ शकते आणि कॉम्प्रमाईज केले जाऊ शकते. रिक्वायरमेंट विविध स्टेकहोल्डर्सकडून आल्या आहेत. ॲम्बीग्यूटी आणि कॉनफ्लिक्ट दूर करण्यासाठी, क्लॅरिटी आणी करेक्टनेस त्यांच्याबद्दल चर्चा केली जाते. अनरियालिस्टिक रिक्वायरमेंटमध्ये तडजोड केली जाते.
4. **डॉक्युमेंटेशन:** सर्व औपचारिक आणि अनौपचारिक, फंक्शनल आणि नॉन-फंक्शनल रिक्वायरमेंट दस्तऐवजीकरण केल्या आहेत आणि पुढील टप्प्यातील प्रक्रियेसाठी उपलब्ध आहेत.

रिक्वायरमेंट एलिसिटेशन थ्रू इंटरव्यूव (Requirement Elicitation through Interview) :

इंटरव्यूव हा समोरासमोर परस्परसंवादासह माहिती गोळा करण्याचा प्राथमिक मार्ग आहे. युजरच्या रिक्वायरमेंटसाठी, माहितीचे कॉलिटेटिव्ह आणि कॉन्टीटेटीव्ह (qualitative and quantitative) दोन्ही प्रकार आवश्यक आहेत. मुलाखती ही बहुतेक वेळा ओपिनियन, पॉलिसीज आणि ॲक्टिव्हिटीज किंवा प्रॉब्लेमचे कथात्मक वर्णन (न्यारेटिव्ह डिस्क्रिप्शन) यासारख्या गुणात्मक माहितीचे सर्वोत्तम माध्यम असतात. बरेच लोक, जे नाखूष आहेत किंवा स्वत: ला लेखनात चांगले व्यक्त करण्यास सक्षम नाहीत, त्यांच्या कल्पनांवर मोकळ्या मनाने तोंडी चर्चा करू शकतात. याउप्पर, वरिष्ठ व्यवस्थापकांच्या मुलाखतीचे वेळापत्रक तयार करणे बहुतेक वेळा प्रश्नावली भरण्यापेक्षा सोपे असते. सिस्टम ॲनालिस्ट एकाच वेळी किंवा गटांमध्ये लोकांची मुलाखत घेऊ शकतात. आवश्यकतेनुसार. मुलाखती एकतर स्ट्रक्चर्ड किंवा अनस्ट्रक्चर्ड असू शकतात.

1. स्ट्रक्चर्ड इंटरव्ह्यूज (Structured interviews) अधिक औपचारिक आणि मानकीकृत (standardized) असतात. मुलाखतीत विचारले जाणारे प्रश्न साधारणपणे आधीच तयार केलेले असतात. स्ट्रक्चर्ड इंटरव्ह्यू (Structured interview) सर्व प्रतिसादकर्त्यांसाठी (respondents) प्रश्नांची एकसमान मांडणी सुनिश्चित करते, ज्यामुळे प्रतिसादांचे (responses) वस्तुनिष्ठ मूल्यमापन (objective evaluation) आणि विश्लेषण (analysis) सुलभ होते. ते व्यवस्थापित करणे सोपे आहे आणि खूप कमी वेळ लागतो. प्रश्न तयार करण्यासाठी काही प्रमाणात विचार आणि प्रयत्न करावे लागत असले तरी, मुलाखती घेण्यासाठी फार कमी कौशल्याची आवश्यकता असते. तथापि, स्ट्रक्चर्ड इंटरव्ह्यू (structured interview) सर्व परिस्थितींसाठी (situations) योग्य नसू शकते. उच्च पातळीची रचना उत्स्फूर्तता

(spontaneity) आणि प्रतिसादांची (responses) स्पष्टता (clarity) प्रतिबंधित करू शकते आणि त्यामुळे काही लोकांना (विशेषतः उच्च अधिकाऱ्यांना) आवडणार नाही. म्हणून, जेव्हा विश्लेषकाला (analyst) विशिष्ट गोष्टींवर (specific items) वस्तुनिष्ठ माहिती (objective information) हवी असते, तेव्हा स्ट्रक्चर्ड इंटरव्ह्यूजना (structured Interviews) प्राधान्य दिले जाते".

2. अनस्ट्रक्चर्ड इंटरव्ह्यू (Unstructured interview) मध्ये मुक्त प्रश्न आणि उत्तर सत्रे वापरली जातात. मोकळे आणि मुक्त वातावरण प्रतिसादकर्त्यांच्या (respondents) भावना, कल्पना आणि विश्वासांबद्दल (feelings, ideas, and beliefs) जाणून घेण्यासाठी अधिक संधी प्रदान करते. मुलाखत घेणारा (interviewer) विस्तृत क्षेत्र कव्हर करू शकतो. मुलाखतीदरम्यानची उत्स्फूर्त चर्चा (spontaneous discussion) दुर्लक्षित राहिलेले किंवा कमी महत्त्वाचे वाटलेले मुद्दे (issues) समोर आणू शकते. मुलाखत घेणारा (interviewer) चांगल्या प्रकारे समजून न घेतलेल्या किंवा अस्पष्ट असलेल्या विशिष्ट मुद्द्यांबद्दल (points) स्पष्टीकरण (clarification) देखील विचारू शकतो. अनस्ट्रक्चर्ड इंटरव्ह्यूजना सेट अप करण्यासाठी कमी वेळ लागतो कारण प्रश्नांची नेमकी मांडणी अगोदरच आवश्यक नसते. तथापि, अनस्ट्रक्चर्ड इंटरव्ह्यूजद्वारे आवश्यक तथ्ये (essential facts) गोळा करण्यासाठी जास्त वेळ लागतो. निष्कर्षांचे (results) विश्लेषण (Analysis) आणि अर्थ लावणे (interpretation) यासाठी देखील स्ट्रक्चर्ड इंटरव्ह्यूपेक्षा (structured interview) जास्त वेळ लागतो. मूल्यमापनाच्या (assessment) व्यक्तिनिष्ठ स्वरूपामुळे (subjective nature), मुलाखत घेणारे (interviewers) निष्कर्ष (results) नोंदवताना त्यांचे पूर्वग्रह (biases) आणू शकतात."

रिक्वायरमेंट एलिसिटेशन थ्रू क्वेश्चनअरी (Requirement Elicitation through Questionnaire):

वेळेच्या अडचणींमुळे, मर्यादित लोकांनाच मुलाखत देता येते. सिस्टम ॲनालिस्टसाठी मोठ्या संख्येने लोकांशी संपर्क साधून सिस्टमच्या विविध पैलूंबद्दल (aspects) त्यांची मते जाणून घेण्यासाठी क्वेश्चनअरी हा एक सोयीस्कर मार्ग आहे. क्वेश्चनअरी हे एक तंत्र आहे, ज्यामध्ये प्रतिसादकर्त्यांकडून (respondents) माहिती आणि मत (opinion) मिळवण्यासाठी (संकलित करण्यासाठी) एका दस्तऐवजाचा (document) वापर केला जातो. क्वेश्चनअरी हा एक दस्तऐवज (document) आहे ज्यात पूर्वनिर्धारित वस्तुनिष्ठ प्रश्न (pre-defined set of objective questions) आणि संबंधित पर्याय (respective options) असतात, जे सर्व स्टेकहोल्डर्सना उत्तरे देण्यासाठी दिले जातात आणि नंतर ते गोळा करून संकलित केले जातात. जेव्हा सिस्टम मोठी असते किंवा अभ्यासाची व्याप्ती अनेक विभागांना कव्हर करते, तेव्हा सिस्टमबद्दल (system) आवश्यक माहिती (necessary facts) देण्यासाठी योग्य असलेल्या सर्व व्यक्तींना क्वेश्चनअरीज मोठ्या प्रमाणात वितरणामुळे (Wide distribution) प्रतिसादकर्त्याला (respondent) अधिक निनावीपणा (anonymity) मिळतो आणि अधिक प्रामाणिक उत्तरांना प्रोत्साहन मिळते. मानकीकृत प्रश्न (Standardized questions) देखील अधिक विश्वासार्ह डेटा (reliable data) देऊ शकतात. तथापि, क्वेश्चननेअरना (questionnaires) पुरेसा प्रतिसाद मिळणे ही अनेकदा एक मोठी समस्या असते. जरी मोठ्या संख्येने व्यक्तींना वितरण केले गेले असले तरी, पूर्ण प्रतिसाद (total response) मिळणे खूप दुर्मिळ आहे."

रेकॉर्ड रिव्हिव (Record Review) :

बऱ्याच संस्था आणि कंपन्यांमध्ये रेकॉर्ड आणि रिव्हिव एक चांगले प्रमाण उपलब्ध असते, जे एक्झिस्टिंग सिस्टीमबद्दल उपयुक्त माहिती प्रदान करतात. 'रेकॉर्ड' हा शब्द रिटन पॉलिसी मॅन्युअल, रेगुलेशन्स आणि स्टँडर्ड ऑपरेटिंग प्रोसिजर्सचा संदर्भ देते जे बहुतेक संस्था व्यवस्थापक आणि कर्मचाऱ्यांसाठी मार्गदर्शक म्हणून राखतात. मॅन्युअल ही कागदपत्रे आहेत जी संस्थेच्या विद्यमान कार्यपद्धती आणि ऑपरेशन्सचे वर्णन करतात तथापि, बहुतेक संस्थांमध्ये उपलब्ध मॅन्युअल आणि मानक ऑपरेटिंग प्रोसेस बऱ्याचदा जुन्या आणि कालबाह्य असतात आणि त्या संस्थांमध्ये त्या सामान्य पद्धतींपेक्षा भिन्न असतात. रेकॉर्ड सिस्टम ॲनालिस्टना सध्याच्या पद्धतींसह परिचित होण्यासाठी इनेबल करतात आणि यामुळे वोल्युम ऑफ ट्रांजेक्शनच्या मात्राबद्दल कल्पना येते. विविध प्रकारचे फॉर्म कसे वापरले जातात याचा काळजीपूर्वक अभ्यास संस्थेमध्ये सध्याच्या पद्धतींचे पालन करण्याबद्दल अधिक चांगले समज प्रदान करते. मागील अभ्यास, सल्लागार संक्षिप्त माहिती आणि व्यवस्थापन अहवालांचे अहवाल देखील विविध बारकाव्यांबद्दल अंतर्दृष्टी प्रदान करतात. हे ॲनालिस्टना समजणे कठीण वाटू शकते अशा काही गोष्टींच्या मागे तर्क प्रदान करते.

ऑब्झर्वेशन (Observation) :

ऑब्झर्वेशनमध्ये ॲनालिस्ट समाविष्ट असतात त्यांचे क्लायंट त्यांची दैनंदिन कामे पार पाडणारे आणि ते काय करीत आहेत आणि का करीत आहेत याबद्दल प्रश्न विचारत आहेत. तज्ञांच्या निरीक्षणाच्या कार्यसंघामध्ये क्लायंटच्या संस्था किंवा कामाच्या ठिकाणी भेट दिली जाते. ते विद्यमान स्थापित सिस्टमचे वास्तविक कार्य निरीक्षण करतात. ते क्लायंटच्या शेवटी वर्कफ्लोचे निरीक्षण करतात आणि अंमलबजावणीच्या समस्येवर कसे कार्य केले जाते. लोक त्यांच्या नोकरीच्या अंमलबजावणीच्या कृतीत निरीक्षण करणे एखाद्या सिस्टमबद्दल तथ्य गोळा करण्याचे एक प्रभावी तंत्र आहे. एखाद्या संस्थेमध्ये केलेल्या विविध उपक्रमांचा अभ्यास करण्यासाठी वैज्ञानिक, समाजशास्त्रज्ञ, मानसशास्त्रज्ञ आणि औद्योगिक अभियंत्यांनी तथ्य शोध (एकत्रित) तंत्र म्हणून निरीक्षण केले आहे. क्रियाकलप प्रत्यक्षात कसे केले जातात याबद्दल निरीक्षण प्रथम-हाताने माहिती प्रदान करते. प्रक्रियेच्या मॅन्युअलमध्ये विहित केलेल्या विशिष्ट चरणांमध्ये विविध क्रियाकलापांमध्ये प्रत्यक्षात पालन केले जाते की नाही हे शोधण्यास विश्लेषकांना सक्षम करते. उदाहरणार्थ, ज्येष्ठ स्तरीय व्यवस्थापकांकडून निर्णय कसे घेतले जातात हे शोधून काढू इच्छित असलेले ॲनालिस्ट ज्या माहितीचा शोध घेण्यात आल्या आहेत त्या प्रकाराचे निरीक्षण करू शकतात, लवकरच या प्रदान केल्या जातात आणि ते कोठून येतात.

प्रॉब्लेम्स इन एलिसिटिंग रिक्वायरमेंटस (Problems in Eliciting Requirements) :

रिक्वायरमेंटसची माहिती देताना बऱ्याच समस्या उद्भवू शकतात, ज्या खाली स्पष्ट केल्या आहेत:

1. **युजर्सना रिक्वायरमेंटसबद्दल खात्री नसते:** ही परिस्थिती बऱ्याचदा उद्भवते जेव्हा सिस्टम डेव्हलपड केली जाण्याची पूर्णपणे नवीन सिस्टीम असते आणि कोणतीही संबंधित मॅन्युअल सिस्टम किंवा कोणतीही मागील सॉफ्टवेअर सिस्टम नसते ज्याचे विश्लेषण आवश्यकतेसाठी (requirement) केले जाऊ शकते.
2. **कॉनफ्लिक्टिंग रिक्वायरमेंट:** प्रकल्पात सामील असलेल्या स्टेकहोल्डर्सच्या संख्येसह ही समस्या वाढते. प्रत्येक स्टेकहोल्डर्सना ते व्यापलेल्या व्यवसाय क्षेत्राच्या आधारे कॉनफ्लिक्टिंग नीड्स आणि प्रायोरिटीज असू शकतात. उदाहरणार्थ, मार्केटिंग टीमचा एक स्टेकहोल्डर रिक्वायरमेंट प्रदान करण्याचा प्रयत्न करेल जे नवीन उत्पादनांच्या निर्मितीस वेगवान करेल, तर सिस्टमचे एंड यूजर्स सुलभ डेटा कॅप्चर करण्यास मदत करणारे इफिसिएंट स्क्रीन्स शोधतील.
3. **व्होलाटाईल रिक्वायरमेंट: रिक्वायरमेंट एलिसिटेशन: (Requirement Elicitation)** स्टेजमध्ये नवीन **स्टेकहोल्डर्स** आल्याने रिक्वायरमेंटमध्ये बदल होऊ शकतात, कारण त्यांचे सिस्टमबद्दल भिन्न दृष्टिकोन (perspectives) असू शकतात. हे बदलांमुळे आवश्यकता (requirements) अधिक चांगल्या प्रकारे स्पष्ट होण्यास मदत होते, तरीही यामुळे कधीकधी रिक्वायरमेंट्स इंजिनियर आणि स्टेकहोल्डर्स यांच्यात सततच्या व्याप्तीतील (scope) बदलामुळे घर्षण (friction) निर्माण होऊ शकते.
4. **कम्युनिकेशन गॅप:** नवीन प्रणाली (new system) विकसित करण्याची गरज अंतिम वापरकर्त्यांना (end users) आणि स्टेकहोल्डर्सना स्पष्ट असली तरी, कंप्यूटिंग सिस्टीम्सच्या (computing systems) कमी एक्सपोजरमुळे त्यांना ती ठोस पद्धतीने (concrete manner) मांडणे कठीण वाटू शकते. ते आवश्यकता (requirements) अस्पष्ट (ambiguous) किंवा न तपासण्यायोग्य (non-testable) पद्धतीने सांगू शकतात.

2.3.2.6 डेव्हलपिंग यूज केसेस (Developing Use Cases) :

यूज केसेस ही एक रिक्वायरमेंटस शोधण्याचे तंत्र आहे जी प्रथम ऑब्जेक्टरी मेथडमध्ये सादर केली गेली (जेकबसन एट अल, 1993). ते आता युनिफाइड मॉडेलिंग भाषेचे फंडामेंटल फिचर बनले आहेत. त्यांच्या सर्वात सोप्या फॉर्ममध्ये, एक यूज केस इंटरॅक्शनमध्ये सामील असलेल्या ॲक्टरना ओळखते आणि इंटरॅक्शन टाईपला नावे देते. त्यानंतर हे अतिरिक्त माहितीद्वारे पूरक आहे (टेक्सटुअल डिस्क्रिप्शन किंवा यूएमएल सीक्वेन्स किंवा स्टेट चार्ट सारख्या एक किंवा अधिक ग्राफिकल मॉडेल्स असू शकतात) सिस्टमशी असलेले इंटरॅक्शनचे वर्णन करतात. यूज केसेस उच्च-स्तरीय यूज केस आकृती वापरुन दस्तऐवजीकरण (documentation) केली जातात. यूज केसेसचा संच सिस्टम रिक्वायरमेंटस मध्ये वर्णन केलेल्या सर्व संभाव्य परस्परसंवादाचे प्रतिनिधित्व करतो. प्रक्रियेतील ॲक्टर, जे इतर सिस्टमचा ॲक्टर असू शकतात, त्यांना स्टिक फिगर म्हणून दर्शविले जाते. इंटरॅक्शनचा प्रत्येक क्लास नामांकित लंबवर्तुळ म्हणून दर्शविला जातो.

रेषा ॲक्टरना इंटरॅक्शनसह जोडतात. वैकल्पिकरित्या, परस्परसंवाद कसा सुरू केला जातो हे दर्शविण्यासाठी एरोहेड्स लाइनमध्ये जोडल्या जाऊ शकतात. हे Fig.2.1 मध्ये दर्शविले गेले आहे जे सेफ होम सिक्युरिटी फंक्शनसाठी काही यूज

केसेस दर्शविते. यूज केसेस सिस्टम आणि त्याचे युजर्स किंवा प्रत्येक वापरलेल्या इतर सिस्टममधील वैयक्तिक संवाद ओळखा प्रत्येक यूज केसचे टेक्सटुअल डिस्क्रिप्शनसह डॉक्युमेंटेशन केले पाहिजे. त्यानंतर हे यूएमएलमधील इतर मॉडेल्सशी जोडले जाऊ शकते जे अधिक तपशीलवार सिनरीओज डेव्हलप करेल.

2.3.2.7 बिल्डिंग द रिक्वायरमेंट मॉडेल (Building the Requirements Model) :

सॉफ्टवेअर इंजीनियरिंगमधील ॲनालिस्ट मॉडेलचा हेतू संगणक-आधारित सिस्टमसाठी रिक्वायर्ड इन्फॉर्मेशन, फंक्शनल आणि बिहेवियरल डोमेनचे वर्णन प्रदान करणे आहे. युजरने तयार होण्याच्या सिस्टमबद्दल अधिक जाणून घेतल्यामुळे ॲनालिस्ट मॉडेल डायनामिकली बदलते आणि इतर स्टेकहोल्डर्स त्यांना खरोखर काय रिक्वायर्ड आहे याबद्दल अधिक समजतात. त्या कारणास्तव, ॲनालिस्ट मॉडेल कोणत्याही वेळी रिक्वायरमेंटसचे स्नॅपशॉट आहे. रिक्वायरमेंटस मॉडेल डेव्हलप होत असताना, विशिष्ट घटक तुलनेने स्थिर होतील, जे पुढील डिझाइन कार्यांसाठी एक ठोस पाया प्रदान करतात. संगणक-आधारित सिस्टमची रिक्वायरमेंटस पाहण्याचे बरेच भिन्न मार्ग आहेत. काही सॉफ्टवेअर लोक असा युक्तिवाद करतात की प्रतिनिधित्वाचा एक मोड निवडणे चांगले आहे (उदा. यूज केस). इतर प्रॅक्टिशनर्सचा असा विश्वास आहे की रिक्वायरमेंटस मॉडेलचे वर्णन करण्यासाठी अनेक वेगवेगळ्या पद्धती वापरणे फायदेशीर आहे. वेगवेगळ्या दृष्टिकोनांकडील रिक्वायरमेंटसचा विचार करण्यासाठी रिप्रेझेंटेशन फोर्स वेगवेगळे प्रकार. सर्वसाधारण घटकांचा एक संच बऱ्याच रिक्वायरमेंटस मॉडेल्समध्ये सामान्य असतो.

1. सिन्यारीओ - बेस्ड एलिमेंट्स (Scenario-based Elements) :

सिन्यारीओ बेस्ड एलिमेंट्स वापरुन युजरच्या दृष्टिकोनातून सिस्टमचे वर्णन केले आहे. उदाहरणार्थ, मूलभूत यूज - केस आणि त्यांचे संबंधित यूज - केस आकृती अधिक विस्तृत टेम्पलेट आधारित यूज - केस मध्ये डेव्हलप होते. रिक्वायरमेंटस मॉडेलचे सिनरीओ बेस्ड एलिमेंट्स बहुतेक वेळा डेव्हलप केलेल्या मॉडेलचा पहिला भाग असतात. तसे, ते इतर मॉडेलिंग एलिमेंट्सच्या निर्मितीसाठी इनपुट म्हणून काम करतात. Fig. 2.3 रिक्वायरमेंटसची माहिती देण्यासाठी आणि यूज - केस वापरुन त्यांचे रिप्रेझेंटेशन करण्यासाठी यूएमएल (UML) ॲक्टिव्हिटी आकृतीचे सिन्यारीओ बेस्ड रिप्रेझेंटेशन दर्शविते.

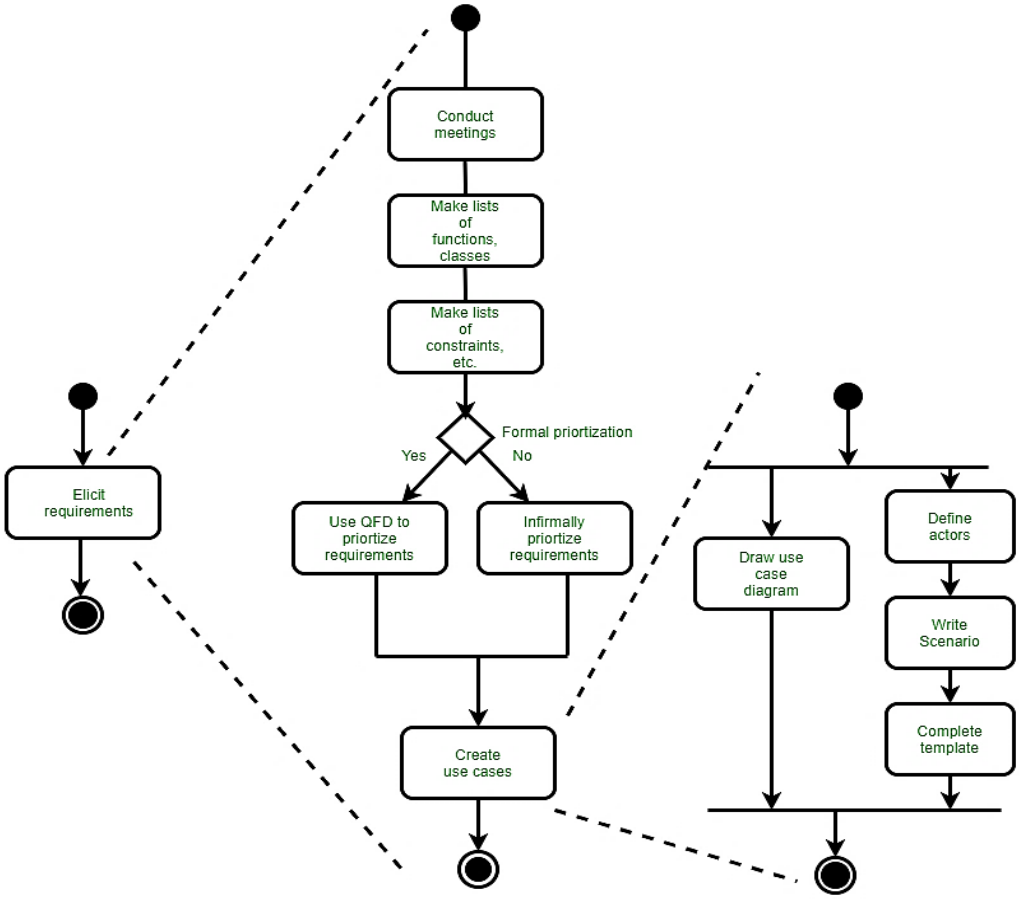


Fig. 2.3: सिन्यारीओ - बेस्ड एलिमेंट्स (Scenario-based Elements)

2.क्लास बेस्ड एलिमेंट्स (Class-based Elements) :

प्रत्येक यूज - केस सिन्यारीओ म्हणजे ॲक्टर म्हणून सिस्टमशी संवाद साधणाऱ्या ॲक्टरच्या रूपात हाताळलेल्या ऑब्जेक्ट्सचा एक संच सूचित करतो. हे ऑब्जेक्ट्स क्लासेस मध्ये वर्गीकृत केले आहेत (सिमिलर ॲट्रिब्यूटस आणि कॉमन बिहेवियर असलेल्या गोष्टींचा संग्रह). उदाहरणार्थ, Fig.2.4 मध्ये दर्शविल्यानुसार सेफहोम सिक्युरिटी फंक्शनसाठी सेन्सर क्लास दर्शविण्यासाठी यूएमएल (UML) क्लास आकृतीचा वापर केला जाऊ शकतो. सेन्सरच्या ॲट्रिब्यूटसची यादी करतो (जसे की नेम, टाईप आणि ऑपरेशन्स सच ॲज आयडेंटिफाय, एनेबल) जे या ॲट्रिब्यूटस सुधारित करण्यासाठी लागू केले जाऊ शकतात.

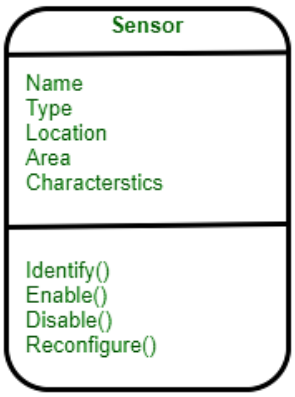


Fig. 2.4: क्लास बेस्ड एलिमेंट्स (Class-based Elements)

3. बिहेवियरल एलिमेंट्स (Behavioral Elements) :

संगणक -आधारित सिस्टमच्या बिहेवियरचा निवडलेल्या डिझाइनवर आणि लागू केलेल्या अंमलबजावणीच्या दृष्टिकोनावर गहन परिणाम होऊ शकतो. म्हणूनच, रिक्वायरमेंटस मॉडेलने मॉडेलिंग एलेमेंट्स प्रदान केले पाहिजेत जे बिहेवियर दर्शविणारे आहेत. स्टेट डायग्राम (State diagram) ही एक पद्धत आहे जी सिस्टमच्या बिहेवियरचे रिप्रेझेंटेशन करते आणि त्याची स्टेट आणि इव्हेंट्स बदलू शकणाऱ्या घटनांचे वर्णन करून स्टेट बदलू शकते. स्टेट हे एक्स्टर्नली ऑब्झरवेबल बिहेवियर आहे. याव्यतिरिक्त, स्टेट डायग्राम एखाद्या विशिष्ट इव्हेंटसाठी परिणामी घेतलेल्या ॲक्शन्स (उदा. प्रोसेस ॲक्टिवेशन) सूचित करते. स्टेट डायग्रामचा वापर स्पष्ट करण्यासाठी, युजरच्या इनपुटला वाचण्यास जबाबदार असलेल्या सेफहोम कंट्रोल पॅनेलमध्ये एम्बेड केलेल्या सॉफ्टवेअरचा विचार करा. एक सिम्पलीफाइड यूएमएल (UML) स्टेट डायग्राम Fig 2.5 मध्ये दर्शविली आहे.

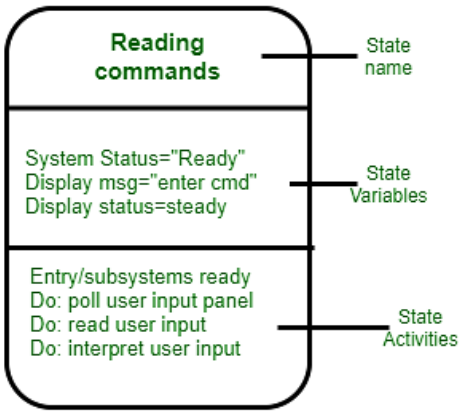


Fig. 2.5 : बिहेवियरल एलिमेंट्स (Behavioral Elements)

4.फ्लो ओरिएंटेड इलेमेंट्स: (Flow-oriented Elements)

संगणक-आधारित सिस्टमद्वारे वाहत असताना माहितीचे रूपांतर होते. सिस्टम विविध प्रकारांमध्ये इनपुट स्वीकारते, त्याचे रूपांतर करण्यासाठी फंक्शन्स लागू करते आणि विविध प्रकारांमध्ये आउटपुट तयार करते. इनपुट हे ट्रान्सड्यूसरद्वारे

प्रसारित केलेले कंट्रोल सिग्नल, मानवी ऑपरेटरद्वारे टाइप केलेल्या संख्येची मालिका, नेटवर्क लिंकवर प्रसारित केलेल्या माहितीचे एक पॅकेट किंवा सेकंडरी स्टोरेजमधून पुनर्प्राप्त केलेली व्हॉल्युमिनस डेटा फाइल असू शकते.

2.3.2.8 नेगोशिएशन (Negotiation) :

जेव्हा एकाधिक स्टेकहोल्डर्सचा सहभाग असतो, रिक्वायरमेंट मध्ये कॉनफ्लीक्ट येतात. ही ॲक्टिव्हिटी रिक्वायरमेंटसला प्रायोरिटी देण्याशी संबंधित आहे आणि नेगोशिएशनद्वारे रिक्वायरमेंटसचे कॉनफ्लीक्ट शोधणे आणि निराकरण करणे. सॉफ्टवेअर टीमवर ठेवलेल्या वास्तविक-जगातील अडचणी (उदा. वेळ, लोक, अर्थसंकल्प) प्रतिबिंबित करताना नेगोशिएशन करण्याचा हेतू म्हणजे स्टेकहोल्डर्सच्या गरजा भागविणारी प्रकल्प योजना डेव्हलप करणे. सर्वोत्कृष्ट ॲक्टिव्हिटी "विन-विन" (win-win) निकालासाठी प्रयत्न करतात म्हणजेच, त्यांच्या बहुतेक गरजा भागविणारी सिस्टीम किंवा प्रॉडक्ट मिळवून स्टेकहोल्डर्स जिंकतात आणि आपण (सॉफ्टवेअर टीमचा मेंबर म्हणून) रियलिस्टिक आणि अचीवेबल बजेट आणि अंतिम मुदतीवर कार्य करून जिंकू शकता. बी. बोहेम (B. Boehm) प्रत्येक सॉफ्टवेअर प्रोसेस इटरेशनच्या सुरुवातीला निगोशिएशन ॲक्टिव्हिटीजचा एक संच परिभाषित करतो. सिंगल कस्टमर कम्युनिकेशन ॲक्टिव्हिटीऐवजी खालील ॲक्टिव्हिटी परिभाषित केले आहेत:

1. सिस्टम किंवा सबसिस्टमच्या मुख्य स्टेकहोल्डर्सची ओळख.
2. डिटर्मिनेशन ऑफ स्टेकहोल्डर्स "विन कंडिशन्स".
3. स्टेकहोल्डर्सच्या नेगोशिएशनने सर्व संबंधित (सॉफ्टवेअर टीमसह) सर्वांसाठी विन-विन कंडिशन्स संचामध्ये समेट करण्यासाठी अटी जिंकल्या.

या प्रारंभिक चरणांची यशस्वी पूर्तता केल्याने विन-विन रिझल्ट प्राप्त होतो, जो त्यानंतरच्या सॉफ्टवेअर अभियांत्रिकी ॲक्टिव्हिटीजकडे जाण्यासाठी मुख्य निकष बनतो.

2.3.2.9 व्हॅलिडेशन (Validation) :

रिक्वायरमेंट्स व्हॅलिडेशन ही एक प्रोसेस आहे, ज्यामध्ये रिक्वायरमेंट्स प्रत्यक्षात कस्टमरला खरोखर हवी असलेली सिस्टम परिभाषित करतात की नाही हे तपासले जाते. ॲनालिसिससोबतच (analysis) यात रिक्वायरमेंट्समधील समस्या शोधण्यावर लक्ष केंद्रित केले जाते. रिक्वायरमेंट्स व्हॅलिडेशन महत्त्वाचे आहे, कारण रिक्वायरमेंट्स डॉक्युमेंटमधील त्रुटींमुळे, या समस्या डेव्हलपमेंट दरम्यान किंवा सिस्टम सेवेत आल्यानंतर आढळल्यास, मोठ्या प्रमाणात पुन्हा काम (extensive rework costs) करावे लागते. रिक्वायरमेंट्स मॉडेलचा प्रत्येक घटक तयार केल्यावर, त्यामध्ये विसंगती (inconsistency), वगळलेले भाग (omissions) आणि संदिग्धता (ambiguity) तपासली जाते. रिक्वायरमेंट्स व्हॅलिडेशन ॲक्टिव्हिटी रिक्वायरमेंट्सची वास्तवता (realism), सुसंगतता (consistency) आणि पूर्णता (completeness) तपासते. मॉडेलद्वारे दर्शविलेल्या रिक्वायरमेंट्सना स्टेकहोल्डर्स प्राधान्य देतात आणि त्या रिक्वायरमेंट्स पॅकेजेसमध्ये (requirements packages) गटबद्ध केल्या जातात, ज्या सॉफ्टवेअर इन्क्रिमेंट्स (software increments) म्हणून अंमलात आणल्या जातील.

रिक्वायरमेंट्स मॉडेलच्या (Requirements model) पुनरावलोकनात (Review) खालील प्रश्नांची उत्तरे दिली जातात:

1. प्रत्येक रिक्वायरमेंट बंधनकारक आणि अस्पष्ट आहे का ?
2. रिक्वायरमेंट मॉडेल "अशा प्रकारे विभाजित केले गेले आहे जे सिस्टमबद्दल क्रमिक अधिक तपशीलवार माहिती उघडकीस आणते ?
3. प्रत्येक रिक्वायरमेंट सिस्टम/ प्रॉडक्टच्या एकूण उद्दीष्टांशी सुसंगत आहे ?
4. ही रिक्वायरमेंट खरोखर आवश्यक आहे की ती सिस्टमच्या उद्देशासाठी आवश्यक नसलेल्या ॲड-ऑन वैशिष्ट्याचे प्रतिनिधित्व करते ?
5. रिक्वायरमेंट मॉडेल सुलभ करण्यासाठी आवश्यकतेचे नमुने वापरले गेले आहेत? सर्व नमुने योग्यरित्या सत्यापित केले गेले आहेत? सर्व नमुने ग्राहकांच्या आवश्यकतांशी सुसंगत आहेत ?
6. काही रिक्वायरमेंट इतर रिक्वायरमेंटसह कॉनफ्लीक्ट करतात ?
7. सर्व रिक्वायरमेंट अमूर्ततेच्या योग्य स्तरावर निर्दिष्ट केल्या आहेत? म्हणजेच, काही रिक्वायरमेंट या टप्प्यावर अयोग्य असलेल्या तांत्रिक तपशीलांची पातळी प्रदान करतात ?

8. प्रत्येक रिक्वायरमेंट चाचणी करण्यायोग्य आहे, एकदा अंमलात आणली गेली आहे ?
9. प्रत्येक रिक्वायरमेंटमध्ये विशेषता असते**?** म्हणजेच, प्रत्येक रिक्वायरमेंटसाठी एक स्त्रोत (सामान्यत: विशिष्ट व्यक्ती) नमूद केला आहे ?
10. रिक्वायरमेंटचे मॉडेल तयार केलेल्या सिस्टमची माहिती, कार्य आणि वर्तन योग्यरित्या प्रतिबिंबित करते ?
11. प्रत्येक रिक्वायरमेंट तांत्रिक वातावरणात साध्य करण्यायोग्य आहे जी सिस्टम किंवा उत्पादनास ठेवेल ?
12. रिक्वायरमेंट मॉडेल भागधारकांच्या गरजेचे अचूक प्रतिबिंब आहे आणि ते डिझाइनसाठी एक ठोस पाया प्रदान करते हे सुनिश्चित करण्यासाठी वरील आणि इतर प्रश्नांची उत्तरे दिली पाहिजेत.

2.4 एस आर एस (Software Requirements Specification)

सॉफ्टवेअर रिक्वायरमेंट डॉक्युमेंट (कधीकधी सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन किंवा एसआरएस म्हटले जाते) सिस्टम डेव्हलपर्सनी काय अंमलात आणले पाहिजे याचे अधिकृत विधान आहे ज्यामध्ये सिस्टमसाठी युजर रिक्वायरमेंट आणि सिस्टम रिक्वायरमेंट तपशीलवार तपशील समाविष्ट केले जावे. हे सॉफ्टवेअर सिस्टमसाठी रिक्वायरमेंट स्पेसिफिकेशन आहे आणि विकसित होणाऱ्या सिस्टमच्या बिहेवियरचे संपूर्ण वर्णन दर्शविते आणि यूजरचे सॉफ्टवेअरशी असलेल्या इंटरॅक्शनचे वर्णन करणारे युज केस चा एक संच समाविष्ट करू शकतो. सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन एक डॉक्युमेंट आहे जे सॉफ्टवेअर कसे करेल याचे वर्णन न करता प्रस्तावित सॉफ्टवेअरने काय करावे हे पूर्णपणे वर्णन केले आहे. रिक्वायरमेंट फेज चे मूलभूत लक्ष्य एसआरएस तयार करणे आहे, जे प्रपोज्ड सॉफ्टवेअरच्या संपूर्ण बिहेवियरचे वर्णन करते. एस आर एस ग्राहकांना त्यांच्या स्वतःच्या गरजा समजण्यास मदत करीत आहे. आय ई ई ई (IEEE) सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन परिभाषित करते, एक डॉक्युमेंट जे सॉफ्टवेअर आणि बाह्य इंटरफेसच्या प्रत्येक आवश्यक रिक्वायरमेंटचे (फंक्शन्स, परफॉर्मन्स, डिझाईन कंसट्रेंन्टस आणि क्वालिटी ॲट्रीब्यूटस) स्पष्ट आणि नेमके वर्णन करते. प्रत्येक रिक्वायरमेंटची व्याख्या अशा प्रकारे केली जाते की त्याची अचिवमेंट एखाद्या निर्धारित पद्धतीद्वारे ऑब्जेक्टिवली व्हेरिफाईड केली जाऊ शकते, उदाहरणार्थ, तपासणी, प्रात्यक्षिक, विश्लेषण किंवा चाचणी"**.**

एस आर एस ची वैशिष्ट्ये (Features of SRS) :

1. हे सॉफ्टवेअर विकासासाठी आधार बनवते.
2. एसआरएस अंतिम सॉफ्टवेअर उत्पादनाच्या व्हॅलिडेशनसाठी एक संदर्भ प्रदान करते.
3. क्लायंटद्वारे मिडीयम किंवा मीडिया आणि वापरलेल्या गरजा अचूकपणे स्पेसिफाईड केल्या आहेत.
4. एसआरएस ग्राहकांना त्यांच्या स्वतःच्या गरजा आणि आवश्यकता समजण्यास मदत करते.
5. हे ग्राहक आणि सप्लायर यांच्यात कराराचा आधार स्थापित करते.

एस आर एस द्वारे सेवा दिलेल्या उद्देशाने (Purposes served by SRS) :

1. **फीडबॅक:** फीडबॅक प्रदान करतो, जो वापरकर्त्यास हे सुनिश्चित करतो की संस्था (जे सॉफ्टवेअर विकसित करते) इशुज किंवा प्रॉब्लेम्सचे निराकरण करण्यासाठी आणि त्या समस्यांचे निराकरण करण्यासाठी आवश्यक सॉफ्टवेअर बिहेवियर समजते.
2. **डीकंपोज प्रॉब्लेम्स इनटू कंपोनंट:** माहितीचे आयोजन करते आणि समस्येचे घटक सुव्यवस्थित पद्धतीने त्याच्या घटक भागांमध्ये विभाजित करते.
3. **व्हॅलिडेशन:** रिक्वायरमेंट योग्यरित्या सांगितल्या गेल्या आहेत हे कबूल करण्यासाठी रिक्वायरमेंटनुसार लागू केलेल्या व्हॅलिडेशन धोरणांचा वापर करते.
4. **इनपुट टू डिझाईन:** डिझाइन सोल्यूशन तयार करण्यासाठी फंक्शनल सिस्टम रिक्वायरमेंटमध्ये पुरेसे तपशील आहेत.
5. **वापरकर्ता बेसिस फोर एग्रीमेंट बिटवीन द युजर अँड द ऑर्गनायझेशन:** सिस्टमद्वारे केलेल्या फंक्शनचे संपूर्ण वर्णन प्रदान करते. याव्यतिरिक्त, निर्दिष्ट केलेल्या रिक्वायरमेंट पूर्ण केल्या आहेत की नाही हे निर्धारित करण्यात वापरकर्त्यांना हे मदत करते.
6. **रिड्युस द डेव्हलपमेंट एफर्ट्स:** सिस्टमची रचना सुरू होण्यापूर्वी विकसकांना वापरकर्त्याच्या रिक्वायरमेंट चा विचार करण्यास सक्षम करते. परिणामी, नंतरच्या टप्प्यात 'रीवर्क' आणि विसंगती कमी केल्या जाऊ शकतात.
7. **इस्टिमेटिंग कॉस्ट अँड शेड्युल:** सिस्टमची रिक्वायरमेंट निश्चित करते आणि अशा प्रकारे विकास प्रकल्पाच्या एकूण किंमतीचा आणि वेळापत्रकांचा 'खडबडीत' अंदाज लावण्यास सक्षम करते.

कॉन्सेप्ट ऑफ एस आर एस (Concept of SRS) :

बाहेरील कंत्राटदार सॉफ्टवेअर सिस्टम विकसित करीत असताना महत्वाची कागदपत्रे आवश्यक असतात. सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन (एस आर एस) सॉफ्टवेअर उत्पादनासाठी रिक्वायरमेंटचे तांत्रिक स्पेसिफिकेशन आहे. एसआरएस सॉफ्टवेअर आवश्यकता व्याख्या क्रियाकलापांच्या निकालाची नोंद करते. एसआरएस देखील रिक्वायरमेंट डॉक्युमेंट म्हणून ओळखले जातात. सॉफ्टवेअर रिक्वायरमेंट डेफिनिशनचे मुख्य लक्ष्य म्हणजे फॉर्मल नोटेशन योग्य म्हणून वापरुन, सॉफ्टवेअर उत्पादनासाठी तांत्रिक रिक्वायरमेंट पूर्णपणे आणि सातत्याने निर्दिष्ट करणे. सॉफ्टवेअर उत्पादनाच्या साईज आणि कॉम्प्लेक्सीटीवर अवलंबून, एसआरएसमध्ये काही पाने असू शकतात आणि ती सिस्टम डेफिनिशनवर आधारित आहे. रिक्वायरमेंट स्पेसिफिकेशन सॉफ्टवेअर उत्पादनाचे "कसे" असे सूचित करतात की "सॉफ्टवेअर डिझाइन कसे आहे हे ठरविल्याशिवाय उत्पादन आवश्यक फीचर्स आणि गोल्स कशी प्रदान करेल हे निर्दिष्ट करते. तथापि, Table: 2.6 मध्ये सादर केलेली सरलीकृत बाह्यरेखा स्पेसिफिकेशनसाठी एक फ्रेमवर्क म्हणून वापरली जाऊ शकते.

1. "इंट्रोडक्शन" मध्ये संगणक-आधारित प्रणालीच्या सामग्रीमध्ये त्याचे वर्णन करणारे सॉफ्टवेअरची गोल्स आणि ऑब्जेक्टिव्हस् आहेत. वास्तविक, हे सॉफ्टवेअरच्या स्कोपशिवाय काही नाही.
2. "इन्फॉर्मेशन डिस्क्रिप्शन" सॉफ्टवेअरला सोडवायच्या समस्यांचे तपशीलवार वर्णन प्रदान करते. इन्फॉर्मेशन कन्टेन्ट आणि रिलेशनशिप, फ्लो आणि स्ट्रक्चर डॉक्युमेंटेड केले आहेत. एक्स्टर्नल सिस्टीम इलेमेंट आणि इंटरनल सॉफ्टवेअर फंक्शन्ससाठी हार्डवेअर, सॉफ्टवेअर आणि ह्यूमन इंटरफेसचे वर्णन केले आहे.
3. समस्येचे निराकरण करण्यासाठी रिक्वायर्ड असलेल्या प्रत्येक फंक्शनचे डिस्क्रिप्शन "फंक्शनल डिस्क्रिप्शन" मध्ये सादर केले आहे. प्रत्येक फंक्शनसाठी एक प्रोसेस न्यारेटीव्ह प्रदान केले जाते, डिझाइनची मर्यादा नमूद केली जाते आणि जस्टीफाइड आहे, परफॉर्मन्स क्यारॅक्टरिस्टीक सांगितली जातात आणि सॉफ्टवेअरच्या संपूर्ण स्ट्रक्चरचे ग्राफिकरित्या प्रतिनिधित्व करण्यासाठी एक किंवा अधिक आकृत्या समाविष्ट केल्या जातात आणि सॉफ्टवेअर फंक्शन्स आणि इतर सिस्टम इलेमेंटमधील इंटरप्ले.
4. स्पेसिफिकेशनचा "बिहेविरल डिस्क्रिप्शन" विभाग बाह्य घटनांचा आणि इंटरनली जनरेट केलेल्या कंट्रोल कॅरेक्टरिस्टिक्सचा परिणाम म्हणून सॉफ्टवेअरच्या ऑपरेशनची तपासणी करतो. कदाचित सर्वात महत्वाचे आणि उपरोधिकपणे, सॉफ्टवेरेक्टिंग स्पेसिफिकेशनचा सर्वात जास्त दुर्लक्षित विभाग म्हणजे "व्हॅलिडेशन क्रायटेरिया". आम्ही यशस्वी इम्प्लिमेंटेशन कशी ओळखू ? फंक्शन, परफॉर्मन्स आणि कॉन्स्टिट्यूएन्टस सत्यापित करण्यासाठी कोणते क्लासेस ऑफ टेस्ट घेतले जाणे आवश्यक आहे ?
5. स्पेसिफिकेशन ऑफ "व्हॅलिडेशन क्रायटेरिया" इतर सर्व रिक्वायरमेंटचे इम्पलीसीट रिव्हीव म्हणून काम करते. या विभागाकडे टाईम आणि अटेंशन देणे आवश्यक आहे.
6. बिब्लीओग्राफी मध्ये सॉफ्टवेअरशी संबंधित सर्व डॉक्युमेंटचे संदर्भ आहेत. यात समाविष्ट आहे: इतर सॉफ्टवेअर इंजीनियरिंग डॉक्युमेंटेशन.टेक्निकल रेफरन्सेस,वेंडर लिटरेचर आणि स्टँडर्ड
7. परिशिष्टात माहिती आहे जी तपशील पूरक आहे. टॅब्युलर डेटा, अल्गोरिदम, चार्ट, आलेख आणि इतर सामग्रीचे तपशीलवार वर्णन परिशिष्ट म्हणून सादर केले आहे

2.4.1 एसआरएसची आवश्यकता/महत्त्व (Need/Importance of SRS) :

एस आर एस साठी खालील बाबींचे वर्णन केले आहे:

1. सिस्टममध्ये तीन प्रमुख पक्ष इच्छुक असतात ते, क्लायंट, युजर्स आणि डेव्हलपर आहेत. ग्राहकांच्या गरजा भागविणारी सिस्टम आणि युजर्सच्या चिंता डेव्हलपरकडे पाठवाव्या लागतील अशा सिस्टमची आवश्यकता आहे. समस्या अशी आहे की क्लायंटला सहसा सॉफ्टवेअर किंवा सॉफ्टवेअर डेव्हलपमेंट प्रोसेस समजत नाही आणि डेव्हलपरला बऱ्याचदा क्लायंटची समस्या आणि ॲप्लीकेशन एरिया समजत नाही. यामुळे डेव्हलपमेंट प्रोजेक्ट मध्ये सामील असलेल्या पक्षांमधील कम्युनिकेशनचे अंतर उद्भवते. सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशन (एसआरएस) चा मूलभूत हेतू हा कम्युनिकेशन मधील अंतर कमी करणे आहे. एसआरएस हे माध्यम आहे ज्याद्वारे क्लायंट आणि युजर्सच्या गरजा अचूकपणे निर्दिष्ट केल्या जातात, खरंच एसआरएस सॉफ्टवेअर डेव्हलपमेंटचा आधार तयार करतो.
2. एसआरएस डेव्हलपकरण्याचा आणखी एक महत्त्वाचा हेतू म्हणजे क्लायंटना त्यांच्या स्वतःच्या नीड आणि रिक्वायरमेंट समजून घेण्यास मदत करणे.

3. एसआरएस महत्त्वपूर्ण आहे कारण ते सॉफ्टवेअर प्रॉडक्ट काय करेल यावर क्लायंट आणि सप्लायर यांच्यात कराराचा आधार स्थापित करते.
4. एक एसआरएस अंतिम प्रॉडक्टच्या व्हॅलिडेशन साठी एक संदर्भ प्रदान करते.
5. हाय क्वालिटीची एसआरएस हाय क्वालिटीचीच्या सॉफ्टवेअरची प्रिरेक्वेसाईट आहे.
6. एसआरएस डेव्हलपर्स सिस्टमच्या डिझाइन करण्यापूर्वी यूजरच्या रिक्वायरमेंटचा विचार करण्यास सक्षम करते. यामुळे पुन्हा काम आणि विसंगती कमी होतात, ज्यामुळे विकासाचे प्रयत्न कमी होते.
7. एसआरएसला फीडबॅक प्रदान करणे आवश्यक आहे, जे वापरकर्त्यास हे सुनिश्चित करते की संस्था इशूज किंवा प्रॉब्लेम्सचे निराकरण करते.
8. एसआरएस सिस्टमच्या रिक्वायरमेंट निश्चित करते आणि अशा प्रकारे डेव्हलपर्सला एकूण किंमतीचा अंदाजे अंदाज लावण्यास आणि सॉफ्टवेअर प्रकल्पाचे वेळापत्रक तयार करण्यास सक्षम करते.
9. एसआरएस रिक्वायरमेंटनुसार योग्यरित्या नमूद केल्या आहेत हे कबूल करण्यासाठी आवश्यकतेनुसार लागू केलेल्या व्हॅलिडेशन स्ट्रॅटेजीसचा वापर करतात.

2.4.2 एस आर एस ची वैशिष्ट्ये (Characteristics of SRS) :

सॉफ्टवेअर रिक्वायरमेंट स्पेसिफिकेशनचा ॲक्युरेट, कम्प्लीट, एफिशियंट आणि हाय क्वालिटीचे असले पाहिजेत जेणेकरून त्याचा संपूर्ण प्रकल्प योजनेवर परिणाम होणार नाही. जेव्हा डेव्हलपर्स आणि यूजरला तयार डॉक्युमेंट सहजपणे समजतात तेव्हा एसआरएस उच्च गुणवत्तेचा असल्याचे म्हटले जाते.

एसआरएसच्या वैशिष्ट्यांविषयी खाली चर्चा केली आहे:

1. **करेक्ट (Correct):** जेव्हा सर्व युजर्सच्या रिक्वायरमेंट डॉक्युमेंटमध्ये नमूद केल्या जातात तेव्हा एसआरएस बरोबर आहे. नमूद केलेल्या रिक्वायरमेंट इच्छित प्रणालीनुसार असाव्यात. याचा अर्थ असा होतो की प्रत्येक रिक्वायरमेंटची तपासणी केली जाते की ते (एसआरएस) युजर्सच्या रिक्वायरमेंटचे प्रतिनिधित्व करते. लक्षात घ्या की एसआरएसच्या करेक्टनेसचे आश्वासन देण्यासाठी कोणतेही निर्दिष्ट टूल किंवा प्रोसिजर नाही. करेक्टनेसहे सुनिश्चित करते की सर्व निर्दिष्ट रिक्वायरमेंट योग्यरित्या केल्या जातात.
2. **अनॲम्बीग्युएस (Unambigous):** जेव्हा प्रत्येक नमूद केलेल्या आ रिक्वायरमेंटचे फक्त एकच स्पष्टीकरण असते तेव्हा एसआरएस इंटरप्रेड आहे. याचा अर्थ असा होतो की प्रत्येक रिक्वायरमेंटचे अनन्य अर्थ लावले जातात. जर एकाधिक अर्थांसह एक संज्ञा वापरली गेली असेल तर, रिक्वायरमेंट डॉक्युमेंट एसआरएस मधील अर्थ निर्दिष्ट केले पाहिजेत जेणेकरून ते स्पष्ट आणि समजण्यास सुलभ असतील.
3. **कम्प्लीट (Complete):** सॉफ्टवेअर काय करणे आवश्यक आहे हे स्पष्टपणे परिभाषित करते तेव्हा एसआरएस पूर्ण होते. यात परफॉर्मन्स, डिझाइन आणि फंक्शनालिटीशी संबंधित सर्व रिक्वायरमेंट समाविष्ट आहेत.
4. **रँकड फॉर इम्पॉर्टन्स स्टॅबिलिटी (Ranked for Inmportance Stability):** सर्व रिक्वायरमेंट तितकीच महत्वाच्या नाहीत, म्हणूनच इतर रिक्वायरमेंटमध्ये फरक करण्यासाठी प्रत्येक रिक्वायरमेंट ओळखली जाते. यासाठी, प्रत्येक रिक्वायरमेंट स्पष्टपणे ओळखणे आवश्यक आहे. स्थिरता भविष्यातील रिक्वायरमेंट बदलांची संभाव्यता सूचित करते.
5. **मॉडीफिएबल (Modifiable):** युजर्सच्या रिक्वायरमेंट बदलू शकतात, म्हणूनच रिक्वायरमेंट डॉक्युमेंट अशा प्रकारे तयार केले जावे की त्या बदलांना सहजपणे सुधारित केले जाऊ शकते, एसआरएसची रचना आणि शैली सातत्याने राखली जाऊ शकते.
6. **ट्रेसेबल (Traceble):** जेव्हा प्रत्येक रिक्वायरमेंटचा स्त्रोत स्पष्ट असेल आणि भविष्यात प्रत्येक रिक्वायरमेंटचा संदर्भ सुलभ करतो तेव्हा एसआरएस शोधण्यायोग्य आहे. यासाठी, फॉरवर्ड ट्रेसिंग आणि बॅकवर्ड ट्रेसिंग वापरले जाते. फॉरवर्ड ट्रेसिंग सूचित करते की प्रत्येक रिक्वायरमेंट डिझाइन आणि कोड इलेमेंटसाठी शोधण्यायोग्य असावी. बॅकवर्ड ट्रेसिंग म्हणजे प्रत्येक रिक्वायरमेंटचे वर्णन स्पष्टपणे त्याच्या सोर्सचा एक्सप्लीसीट रेफरन्स देते.
7. **व्हेरीफिएबल (Verifiable):** अंतिम सॉफ्टवेअर त्या रिक्वायरमेंट पूर्ण करते की नाही हे तपासण्यासाठी स्पेसिफाय केलेल्या रिक्वायरमेंटस व्हेरिफाय केली जाऊ शकतात तेव्हा एसआरएस व्हेरीफिएबल आहे. पुनरावलोकनांच्या मदतीने रिक्वायरमेंट व्हेरिफाय केल्या जातात. लक्षात घ्या की व्हेरीफिएबिलीटीसाठी अनॲम्बीग्यूटी आवश्यक आहे.

8. **कन्सिस्टंट (Consistant) :** जेव्हा परिभाषित केलेल्या वैयक्तिक रिक्वायरमेंटचे सबसेट एकमेकांशी कॉनफ्लीक्ट करीत नाहीत तेव्हा एसआरएस सुसंगत असतात. उदाहरणार्थ, असे एक प्रकरण असू शकते जेव्हा भिन्न रिक्वायरमेंट समान ऑब्जेक्टचा संदर्भ घेण्यासाठी भिन्न अटी वापरू शकतात. निर्दिष्ट आवश्यकता आणि काही आवश्यकतांमध्ये लॉजिकल किंवा टेंपोरल संघर्ष असू शकतात ज्यांची लॉजिकल किंवा टेंपोरल वैशिष्ट्ये समाधानी नाहीत.

2.4.3 एस आर एस फॉरमॅट (SRS Format) :

एसआरएस हे सिस्टम डेव्हलपर्सनी काय अंमलात आणावे याचे स्टँडर्ड विधान आहे. एसआरएसमध्ये सिस्टमसाठी युजर्सच्या रिक्वायरमेंट आणि सिस्टम रिक्वायरमेंटचे स्पेसिफिकेशन समाविष्ट आहेत. एस आर एस हे रिक्वायरमेंट इंजिनीयरद्वारे तयार केलेले अंतिम कार्य उत्पादन आहे. हे त्यानंतरच्या सॉफ्टवेअर इंजिनिअरिंग ॲक्टिव्हिटीजचा पाया म्हणून काम करते. एस आर एस सॉफ्टवेअरच्या फंक्शन आणि परफॉर्मन्स रिक्वायरमेंट चे वर्णन करते. हे त्याच्या विकासावर परिणाम करणारे निर्बंध देखील सूचीबद्ध करते. एसआरएस दस्तऐवजाची एक विशिष्ट रचना (टेम्पलेट) Fig: 2.6 मध्ये दिली आहे. या टेम्पलेटच्या प्रत्येक शीर्षक आणि उप-शीर्षकाच्या विरूद्ध सॉफ्टवेअरच्या रिक्वायरमेंटचे वर्णन केले जाऊ शकते.

Table 2.1 स्ट्रक्चर फॉरमॅट ऑफ एस आर एस (Structure Format of SRS)

सेक्शन 1: प्रॉडक्ट ओव्हरविव आणि समरी
सेक्शन 2: डेव्हलपमेंट, ऑपरेटिंग, अँड मेंटेनन्स एनवोर्मेन्ट
सेक्शन 3: एक्स्टर्नल इंटरफेस अँड डेटा फ्लो
सेक्शन 4: फंक्शनल रिक्वायरमेंन्टस
सेक्शन 5: परफॉर्मन्स रिक्वायरमेंन्टस
सेक्शन 6: एक्सेप्शन हँडलिंग
सेक्शन 7: अरली सबसेट्स अँड इम्प्लिमेंटेशन प्रायोरिटीज
सेक्शन 8: फॉरिसिबल मोडिफिकेशन्स अँड इनहान्समेंट
सेक्शन 9: ॲक्सेप्टन्स क्रायटेरिया
सेक्शन 10: डिझाईन हिन्टस अँड गाईडलाईन्स
सेक्शन 11: क्रॉस रेफरन्स इंडेक्स
सेक्शन 12: ग्लोसरी ऑफ टर्म्स

एस आर एस एक औपचारिक दस्तऐवज आहे. हे नैसर्गिक भाषा, ग्राफिकल प्रतिनिधित्व, गणिताचे मॉडेल, वापर परिदृश्य, प्रोटोटाइप मॉडेल किंवा विकसित केलेल्या सॉफ्टवेअरचे वर्णन करण्यासाठी या कोणत्याही संयोजनाचा वापर करते. सुसंगत आणि अधिक समजण्यायोग्य पद्धतीने आवश्यक वैशिष्ट्ये सादर करण्यासाठी मानक टेम्पलेट्स आहेत. एसआरएस दस्तऐवज सुप्रसिद्ध असावे. एक संरचित दस्तऐवज समजणे आणि सुधारित करणे सोपे आहे. आवश्यकता दस्तऐवज अशा प्रकारे तयार केले गेले आहे जे लिहिणे, पुनरावलोकन करणे आणि देखभाल करणे सोपे आहे. हे स्वतंत्र विभागांमध्ये आयोजित केले जाते आणि प्रत्येक विभाग मॉड्यूल किंवा युनिट्समध्ये आयोजित केला जातो. Fig. 2.6 एसआरएस दस्तऐवजाची रचना दर्शविते.

एस आर एस फॉरमॅटच्या विभागांचे खाली वर्णन केले आहे:

सेक्शन 1 आणि 2 एस आर एस दस्तऐवजाचा प्रॉडक्ट वैशिष्ट्यांचे विहंगावलोकन सादर करतात आणि सॉफ्टवेअर प्रॉडक्टच्या डेव्हलपमेंट, ऑपरेशन्स आणि मेंटेनन्स यासाठी प्रोसेस एनवोर्मेन्टचा सारांश देतात.

सेक्शन 3 एसआरएस डॉक्युमेंटचा सॉफ्टवेअर उत्पादनाची बाह्यरित्या निरीक्षण करण्यायोग्य वैशिष्ट्ये निर्दिष्ट करतो आणि त्यात युजर डिस्प्लेज आणि रिपोर्ट फॉरमॅटस, युजरच्या आदेशांचा सारांश आणि अहवाल पर्याय, डेटा फ्लो डायग्राम आणि डेटा डिक्शनरी समाविष्ट आहे.

सेक्शन 4 एसआरएस डॉक्युमेंटचा सॉफ्टवेअर उत्पादनासाठी फंक्शनल रिक्वायरमेंट निर्दिष्ट करतो. थोडक्यात, फंक्शनल रिक्वायरमेंट रिलेशनल आणि स्टेट-ओरिएंटेड नोटेशनमध्ये व्यक्त केल्या जातात ज्या इनपुट, क्रिया आणि आउटपुट इत्यादींमधील संबंध निर्दिष्ट करतात .

सेक्शन 5 एसआरएस डॉक्युमेंटचा विविध ॲक्टिव्हिटीजसाठी रिस्पॉन्स टाईम, विविध प्रक्रियेसाठी प्रोसेस वेळ, थ्रूपूट, प्राथमिक आणि दुय्यम मेमरी मर्यादा, आवश्यक दूरसंचार बँडविड्थ आणि असामान्य विश्वसनीयता आवश्यकता इ. सारख्या कामगिरीची वैशिष्ट्ये निर्दिष्ट करते.

सेक्शन 6 एसआरएस डॉक्युमेंटमध्ये अपवाद हाताळणी निर्दिष्ट केली आहे, ज्यात ॲक्शन्स करण्याच्या कृती आणि अनडिझायर्ड सिच्युएशन्स किंवा इव्हेंट किंवा एररच्या प्रतिसादात प्रदर्शित केले जाणारे संदेश समाविष्ट आहेत. संभाव्य अपवादांच्या विविध श्रेणींमध्ये तात्पुरते आणि कायमस्वरुपी संसाधन अपयश, चुकीचे, विसंगत किंवा बाहेरील इनपुट डेटा, क्षमतेच्या मर्यादेचे उल्लंघन आणि ऑपरेटरवरील निर्बंधांचे उल्लंघन इ. समाविष्ट आहे.

सेक्शन 7 एसआरएस डॉक्युमेंटच्या मध्ये डेव्हलपमेंट अंतर्गत सिस्टीमसाठी लवकर उपसंच आणि अंमलबजावणीची प्राथमिकता निर्दिष्ट केली आहे आणि विविध सिस्टम क्षमतांसाठी अंमलबजावणीचे प्राधान्यक्रम निर्दिष्ट करणे महत्वाचे आहे.

सेक्शन 8 एसआरएस डॉक्युमेंटचा मध्ये प्रारंभिक उत्पादनांच्या प्रकाशनानंतर उत्पादनात समाविष्ट केलेल्या अंदाजे बदल आणि संवर्धने निर्दिष्ट केल्या आहेत.

सेक्शन 9 एसआरएस डॉक्युमेंटचा सॉफ्टवेअर प्रॉडक्ट ॲक्सेप्टन्स क्रायटेरिया निर्दिष्ट करतो. ॲक्सेप्टन्स क्रायटेरिया फंक्शनल आणि परफॉर्मन्स चाचण्या निर्दिष्ट करतात ज्या केल्या पाहिजेत आणि सोर्स कोड, अंतर्गत डॉक्युमेंटेशन आणि बाह्य डॉक्युमेंटेशन यासारख्या डिझाईन स्पेसिफिकेशन, टेस्ट प्लॅन, युजर्स मॅन्युअल, ऑपरेशन्स प्रिन्सिपल आणि इन्स्टॉलेशन व मेंटेनन्स प्रोसिजर इत्यादी.

सेक्शन 10 एसआरएस डॉक्युमेंटचा डिझाइनची सूचना आणि मार्गदर्शक तत्त्वे निर्दिष्ट करते. उत्पादन अंमलबजावणीचा "कसा" हा सॉफ्टवेअर डिझाइनचा विषय आहे आणि उत्पादनाच्या डिझाइन टप्प्यात पुढे ढकलला पाहिजे.

सेक्शन 11 एसआरएस डॉक्युमेंटचा रिक्वायरमेंट पूर्ण करण्यासाठी वापरल्या जाणाऱ्या माहितीच्या स्त्रोतास उत्पादनांच्या रिक्वायरमेंट निर्दिष्ट करतो.

सेक्शन 12 एसआरएस डॉक्युमेंट मध्ये डेफिनेशन ऑफ टर्म्स निर्दिष्ट केली आहे जी ग्राहक आणि उत्पादन विकसकांना अपरिचित असू शकते. स्टँडर्ड टर्म्स परिभाषित करण्यासाठी योग्य काळजी घेतली पाहिजे ज्या नॉनस्टँडर्ड मार्गांनी वापरल्या जातात.

एस आर एस चे फायदे (Advantages of SRS) :

1. सॉफ्टवेअर एस आर एस क्लायंट आणि सप्लायर यांच्यात सॉफ्टवेअर उत्पादन काय करेल यावर करारासाठी मूलभूत स्थापित करते.
2. एस आर एस अंतिम उत्पादनाच्या प्रमाणीकरणासाठी एक संदर्भ प्रदान करते.
3. उच्च-गुणवत्तेचा एस आर एस उच्च-गुणवत्तेच्या सॉफ्टवेअरची पूर्व शर्त आहे.
4. एक उच्च-गुणवत्तेचा एस आर एस डेव्हलपमेंट कॉस्ट कमी करते.

References:

1. Software Engineering: A practitioner's approach by Roger S. Pressman & Bruce R. Maxim, McGraw Hill Higher Education, New Delhi, (Ninth Edition) ISBN 93-5532-504-5
2. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
3. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi - 2001, ISBN-13: 9780074631218

Websites:

1. https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/
2. https://www.tutorialspoint.com/software_engineering/index.htm
3. https://app.diagrams.net/

E-learning material (Video lectures):

1. https://www.youtube.com/watch?v=WjwEh15M5Rw

युनिट-3
सॉफ्टवेअर मॉडेलिंग आणि डिझाइन
(Software Modelling and Design)

विषय निष्पत्ती (Course Outcome):

CO3: वेगवेगळे सॉफ्टवेअर डिज़ाइन मॉडेल तयार करा.

घटक निष्पत्ती (Theory Learning Outcome - TLO) :

1. दिलेल्या सॉफ्टवेअर आवश्यकतांसाठी ॲनालिसिस मॉडेल चे घटक ओळखा.
2. सॉफ्टवेअर रिक्वायरमेंट्स मॉडेलिंगसाठी निर्दिष्ट डिझाइन (Design Concepts) लागू करा.
3. डिज़ाइन नोटेशन वापरून सॉफ्टवेअर डिज़ाइन तयार करा.
4. सॉफ्टवेअर टेस्टिंगचा उद्देश सांगा.
5. सॉफ्टवेअर प्रकल्पासाठी युज- केस डायग्रॅम , क्लास डायग्रॅम, सिकवेनस डायग्रॅम काढा.
6. सॉफ्टवेअर टेस्टिंगचे मूलभूत प्रकार स्पष्ट करा.

3.1 ट्रान्सलेशन मॉडेल चे डिज़ाइन मॉडेल मध्ये रूपांतर करणेडेटा मॉडेलिंग : (Translating Requirement model into design model: Data Modelling)

डिज़ाइन मॉडेल्सचे ट्रान्सलेशन मॉडेल्स मध्ये रूपांतर करणे हे सॉफ्टवेअर अभियांत्रिकीमधील एक महत्त्वपूर्ण टप्पा दर्शविते जिथे उच्च-स्तरीय सिस्टम (High Level System) हे ट्रान्सलेशन डिटेल्ड, अंमलबजावणी योग्य डिज़ाइनमध्ये रूपांतरित केल्या जातात. हे सर्वसमावेशक अन्वेषण या परिवर्तन प्रक्रियेत समाविष्ट मूलभूत संकल्पना, कार्यपद्धती आणि सर्वोत्तम पद्धतींची तपासणी करते, ट्रान्सलेशन-टू-डिज़ाइन रूपांतर म्हणून मुख्य घटक म्हणून डेटा मॉडेलिंगवर विशेष भर देते.

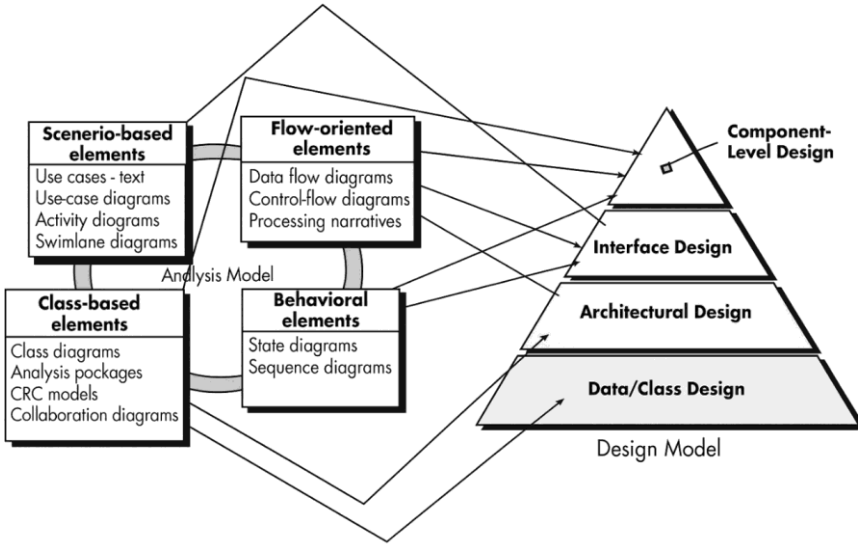


Fig 3.1: ट्रान्सलेशन मॉडेल चे डिज़ाइन मॉडेल मध्ये रूपांतर करणेडेटा मॉडेलिंग : (Translating Requirement model into design model: Data Modelling)

3.1.1 ट्रान्सलेशन प्रोसेस समजून घेणे (Understanding the Translation Process)

डिज़ाइन मॉडेल्समध्ये ट्रान्सलेशन मॉडेल्सचे रूपांतर करताना, एक ॲब्स्ट्रॅक्ट, उच्चस्तरीय- सिस्टम (high-level System) स्पेसिफिकेशन ठोस (concrete) डिज़ाइन ब्लूप्रिंटमध्ये रूपांतरित करावे लागते हे ब्लूप्रिंट नंतर प्रत्यक्ष . अंमलबजावणीसाठी मार्गदर्शक ठरते .ही प्रोसेस म्हणजे सिस्टम काय करणार आहे आणि ते कसे करणार (ट्रान्सलेशन) (डिज़ाइन) आहे यांच्यातला **ब्रिज**((Bridge म्हणून काम करते. ट्रान्सलेशन मॉडेल्स विविध प्रकारांनी प्रकट होतात जसे कि सिनॅरिओ आधारित (scenario-based), क्लास आधारित (class-based), फ्लो आधारित (Flow-oriented) घटक आणि बिहेवीरेल आधारित घटक (Behavioral elements) हे मॉडेल्स थेट डिज़ाइन च्या कामात उपयोगी पडतात. या

प्रक्रियेदरम्यान सॉफ्टवेअर अभियंत्यांनी संपूर्ण आर्किटेक्चरचे बारकाईने ॲनालिसिस करावे लागते त्यानंतर .गरजांनुसार योग्य स्ट्रक्चरल घटक)components) निवडावे लागतात .ही प्रोसेस सहसा डिझाइन घटक (Design Elements) ओळखण्यापासून सुरू होते हे घटक म्हणजे .क्लासेस , इंटरफेस , मॉड्यूल्स आणि इतर सिस्टम घटक (System Elements) हे घटक गरजांनुसार काळजीपूर्वक निवडून आणि संघटित करून एकसंध (cohesive) डिज़ाइन((Design तयार केले जातेहे डिज़ाइन कार्यात्मक .(Functional) आणि गैरकार्यात्मक-(Non Functional) गरजा पूर्ण करू शकते. यासाठी उच्चस्तरीय गरजा अधिक डिटेल्ड-Detailed) डिझाइन स्पेसिफिकेशन्समध्ये परिष्कृत केल्या जातातयात . :बऱ्याचदा

1. जटिल गरजांचे छोटे व सुलभ व्यवस्थापनयोग्य घटकांमध्ये विभाजन केले जाते.
2. हे घटक अंमलबजावणी दरम्यान स्वतंत्रपणे हाताळता येतात.

सॉफ्टवेअर डिज़ाइन) मध्ये सहसा पुनरावृत्ती दृष्टिकोन(Design)iterative approach) वापरला जातो:

1. सुरुवातीला, डिज़ाइन उच्च पातळीवर तयार केले जाते, जिथे ते सिस्टमच्या उद्दिष्टांशी व गरजांशी जोडलेले असते.
2. पुढील पुनरावृत्तीमध्ये डिज़ाइन अधिक डिटेल्ड होत जाते.
3. शेवटी, हे डिज़ाइन इतके परिष्कृत होते की ते थेट अंमलबजावणीसाठी तयार असते.

3.1.2. डेटा मॉडेलिंग फंडामेंटल (Data Modelling Fundamentals)

डेटा मॉडेलिंग डेटाबेस डिज़ाइन प्रक्रियेतील मूलभूत पायरी (Step) आहे आणि एक उच्च-स्तरीय, ऍब्स्ट्रॅक्शन डिज़ाइन टप्पा म्हणून कार्य करते ज्यास बहुतेकदा कॉन्सेप्च्युअल डिज़ाइन म्हणून संबोधले जाते. डेटा मॉडेलिंगचा प्राथमिक उद्देश माहिती चे(Information) स्ट्रक्चर्ड प्रतिनिधित्व तयार करणे आहे जे डेटा बिंदू (Point) आणि संरचनांमधील कनेक्शन स्पष्टपणे रेखाटतात. या प्रक्रियेत डेटा प्रकार, नातेसंबंध (Relationships) आणि मर्यादा (Constraint) निश्चित करणे समाविष्ट आहे जे सिस्टम जीवनचक्रात माहिती (Information) साठविण्यासाठी आणि हाताळण्यासाठी वापरले जातील. डेटा मॉडेलिंगचे मूलभूत उद्दिष्ट म्हणजे सिस्टम मध्ये वापरल्या जाणाऱ्या आणि संग्रहित करण्यात येणाऱ्या डेटाचे प्रकार स्पष्ट करणे. यामध्ये डेटा प्रकारांमधील संबंध (Relationships) समजावून सांगणे, डेटाचे गट तयार करणे आणि संघटित करण्याचे मार्ग ठरवणे, डेटा संरचना (Data Structures) कशी असावी हे ठरवणे, तसेच त्याचे स्वरूप (Format) आणि वैशिष्ट्ये (Characteristics) स्पष्ट करणे, या बाबींचा समावेश होतो. डेटा(Data) मॉडेल्स नेहमीच विशिष्ट व्यावसायिक गरजांभोवती (Business Needs) तयार केली जातात. यासाठी व्यावसायिक भागधारकांचा (Business Stakeholders) अभिप्राय घेतला जातो. त्यांच्या अभिप्रायाच्या आधारे नियम आणि गरजा स्पष्ट केल्या जातात. हे नियम आणि गरजा नवीन सिस्टम डिज़ाइनमध्ये समाविष्ट करता येतात किंवा विद्यमान सिस्टमच्या सुधारित आवृत्तीसाठी (Iteration/Enhancement) अनुकूल करता येतात. डेटा मॉडेलिंग म्हणजे एखाद्या ऑर्गनायझेशनमध्ये डेटा संसाधने कशी परिभाषित (Define) करायची आणि कशी व्यवस्थापित (Manage) करायची यासाठी एक सामान्य, सुसंगत आणि सोपी पद्धत तयार करणे. यासाठी प्रमाणित स्कीमा (Standardized Schema) आणि औपचारिक तंत्र (Formal Techniques) वापरले जातात. सॉफ्टवेअर अभियंते सर्वात जास्त एंटिटी-रिलेशनशिप डायग्राम (Entity Relationship Diagram) या डेटा मॉडेलचा उपयोग करतात. हे मॉडेल डेटाचे प्रतिनिधित्व (Representation) कसे करायचे यावर लक्ष केंद्रित करते आणि एखाद्या अनुप्रयोगात (Application) डेटा(Data) कसा प्रविष्ट (Insert), संग्रहित (Stored), रूपांतरित (Conversion) आणि तयार केला जातो हे दाखवते. ही मॉडेल्स लिव्हिंग डॉक्युमेंट म्हणून काम करतात, म्हणजेच व्यवसायाच्या बदलत्या गरजांनुसार ती सतत सुधारली जातात. या मॉडेल्समुळे व्यवसाय प्रोसेस अधिक चांगल्या प्रकारे चालतात आणि आयटी आर्किटेक्चर आणि रणनीती (Strategy) तयार करण्यात मोठी मदत होते.

3.1.3. डेटा मॉडेलिंग घटक आणि संकल्पना (Data Modelling Elements and Concepts)

डेटा मॉडेलिंगमध्ये अनेक प्रमुख घटकांचा समावेश आहे जे सिस्टम माहिती (Information) आवश्यकतांचे व्यापक प्रतिनिधित्व तयार करण्यासाठी एकत्र काम करतात. डेटा ऑब्जेक्ट्स (Objects) (Data Objects) संमिश्र माहिती चे(Information) प्रतिनिधित्व करतात जे सॉफ्टवेअरद्वारे समजून घेणे आवश्यक आहे, याचा अर्थ असा आहे की ज्यात अनेक भिन्न गुणधर्म किंवा वैशिष्ट्ये आहेत. उदाहरणार्थ, केवळ रुंदी (width) वैध डेटा ऑब्जेक्ट तयार करणार नाही, उंची (Height), रुंदी (Width) आणि खोली (Depth) समाविष्ट करणारी परिमाणे एकाधिक वैशिष्ट्यांसह संपूर्ण ऑब्जेक्ट म्हणून परिभाषित केली जाऊ शकतात.

या प्रक्रियेचे उद्दीष्ट (Aim) तीन मूलभूत पैलूंचे वर्णन करणे आहे: डेटा बेसमध्ये समाविष्ट डेटा जसे की विद्यार्थी (students), व्याख्याते (lecturers), अभ्यासक्रम (courses) आणि विषय (subjects) यासारख्या संस्था, डेटा आयटममधील संबंध जसे की अभ्यासक्रम (courses) शिकविणारे व्याख्याते (lecturers) किंवा व्याख्याते (lecturers) देखरेख करतात, आणि डेटावरील मर्यादा जसे की आठ अंकी (Eight Digit) विद्यार्थी संख्या किंवा केवळ चार किंवा सहा युनिट क्रेडिट असलेले विषय. हे घटक एकत्रितपणे समस्येचे निराकरण करण्यासाठी माहिती डोमेनची (information domain) व्यापक समज तयार करण्यासाठी कार्य करतात. ॲनालिसिस क्लासेस डेटा मॉडेलिंग संदर्भात विविध स्वरूपात प्रकट होतात. यामध्ये इतर सिस्टम, उपकरणे किंवा संगणक-आधारित सिस्टमसाठी माहिती (Information) तयार करणारे किंवा वापरणारे लोक (Users) यासारख्या बाह्य संस्थांचा (external entities) समावेश आहे; अहवाल (reports), प्रदर्शने (displays), पत्रे (letters) किंवा सिग्नल यासारख्या गोष्टी ज्या माहिती (Information) डोमेनचा भाग आहेत; मालमत्ता हस्तांतरण किंवा सिस्टम ऑपरेशनमध्ये होणार्या रोबोट हालचाली पूर्ण होण्यासारख्या घटना; व्यवस्थेशी संवाद साधणार् या लोकांनी बजावलेल्या भूमिका; अनुप्रयोगाशी संबंधित संस्थात्मक युनिट्स; समस्या संदर्भ स्थापित करणारी ठिकाणे; आणि ऑब्जेक्ट्स किंवा संबंधित ऑब्जेक्ट क्लासेसचे क्लास परिभाषित करणारी रचना.

3.1.4 डेटा मॉडेलचे प्रकार (Types of Data Models)

डेटा मॉडेलिंग ही प्रोसेस ऍब्स्ट्रॅक्शनच्या विविध स्तरांवर काम करते. ती उच्च-स्तरीय कॉन्सेप्च्युअल प्रतिनिधित्वापासून सुरू होते आणि हळूहळू डिटेल्ड फिज़िकल अंमलबजावणीपर्यंत प्रगती करते. या प्रक्रियेत सामान्यतः तीन वेगवेगळ्या प्रकारची मॉडेल्स असतात: कॉन्सेप्च्युअल , लॉजिकल आणि फिज़िकल मॉडेल्स. प्रत्येक प्रकाराचे उद्दिष्ट वेगळे असते आणि संपूर्ण विकास जीवनचक्रात (Development Life Cycle) विशिष्ट प्रेक्षकांसाठी तयार केले जाते.

3.1.4.1 कॉन्सेप्च्युअल मॉडेल्स (Conceptual Models)

ही व्यवसायाच्या आवश्यकतांमधून गोळा केलेल्या ऍब्स्ट्रॅक्शनआणि मॉडेलमाहितीच्या(Information) सर्वोच्च पातळीचे प्रतिनिधित्व करतात. यात डेटाबेस डिझाइनच्या तांत्रिक बाबींचा विचार केला जात नाही. कॉन्सेप्च्युअल ईआरडी (Conceptual ERD) मध्ये संस्था (Entities) आणि संबंध (Relationships) केवळ व्यावसायिक गरजांभोवती परिभाषित केले जातात. कॉन्सेप्च्युअल ईआरडी (Conceptual ERD) हे सर्व प्रकारच्या मॉडेल्स पैकी सर्वात सोपे असतात आणि संस्थांमधील संबंध मॉडेलिंग करण्यासाठी सामान्यीकरणाच्या (Generalization) वापराला समर्थन देतात. उदाहरणार्थ, "त्रिकोण (Triangle) हा एक प्रकारचा आकार (Shape) आहे" हे दाखविणे. या मॉडेल्समध्ये "ही माहिती (Information) कशी अंमलबजावीत केली जाईल" यापेक्षा "कोणती माहिती (Information) आहे" यावर जास्त लक्ष दिले जाते.

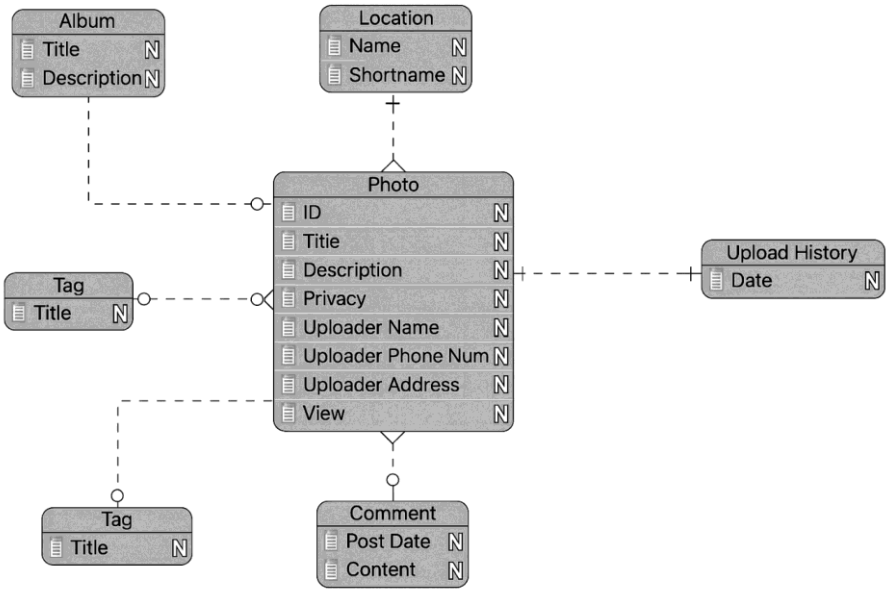


Fig. 3.2: कॉन्सेप्च्युअल मॉडेल्स (Conceptual Models)

3.1.4.2 लॉजिकल मॉडेल्स (Logical Models)

देखील व्यावसायिक आवश्यकतांमधून तयार केली जातात, पण त्या कॉन्सेप्च्युअल मॉडेल्स (Conceptual Models) पेक्षा थोडी अधिक गुंतागुंतीची असतात. कारण यात स्तंभ प्रकार (Column Types) व्यवसाय विश्लेषणाला मदत करण्यासाठी सेट केले जाऊ शकतात. लॉजिकल मॉडेल्समध्ये (Logical Models) स्तंभ प्रकार (Column Types) सेट करणे ऐच्छिक असते आणि जेव्हा ते वापरले जाते तेव्हा ते डेटाबेस(Data)(Database) तयार करण्यासाठी नव्हे, तर व्यवसाय विश्लेषणासाठी उपयोगी असले पाहिजे. लॉजिकल मॉडेल्स(Models) उच्च-स्तरीय व्यवसाय संकल्पना (business concepts) आणि तांत्रिक अंमलबजावणी(technical implementation) विचार यांच्यातील दरी कमी करण्याचे काम करतात.

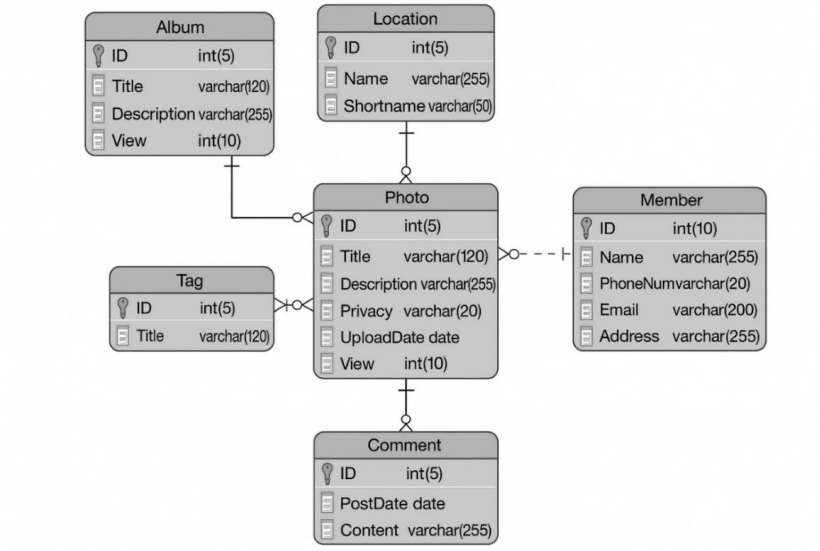


Fig. 3.3: लॉजिकल मॉडेल (Logical Model)

3.1.4.3 फिजिकल मॉडेल्स (Physical Models)

हे रिलेशनल डेटाबेसच्या(Data) प्रत्यक्ष डिज़ाइन(Design) ब्लूप्रिंटचे प्रतिनिधित्व करतात. यात हे दाखवले जाते की विशिष्ट डेटाबेस(Data) मॅनेजमेंट सिस्टम(Database Management System) मध्ये डेटा(Data) कसा रचित (Structured) असावा आणि एकमेकांशी संबंधित असावा. त्यामुळे फिजिकल ईआरडी (Physical ERD) तयार करताना निवडलेल्या DBMS च्या परंपरा (Conventions) आणि निर्बंधांचा (Constraints) विचार करणे खूप महत्त्वाचे असते. फिजिकल मॉडेल्समध्ये(Models) डेटा(Data) प्रकारांचा अचूक वापर करणे आवश्यक असते आणि ही मॉडेल्स(Models) अंमलबजावणीसाठी तयार असलेल्या डेटाबेस(Data)(Database) संरचनेचे प्रत्यक्ष प्रतिनिधित्व करतात.

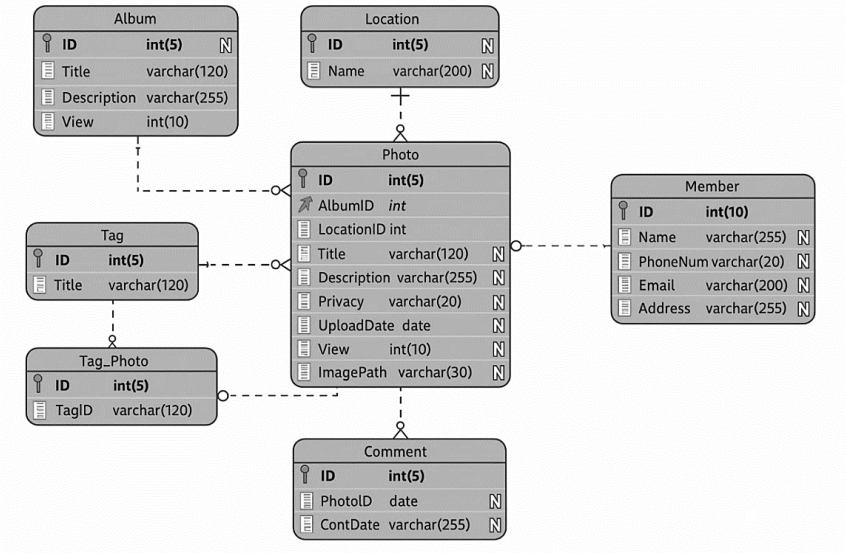


Fig. 3.4: फिजिकल मॉडेल्स (Physical Models)

3.1.5 रिक्वायरमेंट्स कलेक्शन एंड एनालिसिस (Requirement Collection And Analysis)

डेटाबेस(Data) डिझाइन प्रोसेस (Database Design Process) ट्रान्सलेशन(Translation), संग्रह (Collection) आणि ॲनालिसिस(Analysis) यापासून सुरू होते. यामध्ये डेटाबेस(Data) डिझाइनर(Database Designers) संभाव्य डेटाबेस(Data) वापरकर्त्यांशी(Database Users) मुलाखती घेतात, त्यांच्या डेटा(Data) गरजा समजून घेण्यासाठी आणि त्या व्यवस्थित डॉक्यूमेंट (Document) करण्यासाठी. या टप्प्याचा परिणाम म्हणजे वापरकर्त्याच्या आवश्यकतांचा(Users Requirement) एक संक्षिप्त आणि लिहिलेला संच(set) तयार होतो, ज्यामध्ये सिस्टमने(System) डेटा(Data) कसा हाताळावा याबद्दलची सर्व आवश्यक माहिती(Information) असते. डेटा(Data) ट्रान्सलेशन(Data Translation) निश्चित करतानाच, कार्यात्मक ट्रान्सलेशन(Translation)सुद्धा ओळखले पाहिजे. यात वापरकर्त्यांनी परिभाषित केलेली ऑपरेशन्स(Operations) किंवा व्यवहार समाविष्ट असतात, जे डेटा(Data) पुनर्प्राप्ती आणि अद्यतन (Update) ऑपरेशन्ससाठी डेटाबेसवर(Data) लागू केले जातात. गरजांचे ॲनालिसिस (Requirements Analysis) करताना, "ही उद्दिष्टे कशी साध्य होतील" यापेक्षा "सिस्टमने(System) काय केले पाहिजे" यावरच जास्त लक्ष दिले जाते. यात हे समजून घेतले जाते की विशिष्ट परिस्थितीत वापरकर्त्याचा परस्परसंवाद काय असेल, सिस्टम(System) कोणत्या ऑब्जेक्ट्समध्ये(Object) बदल करेल, सिस्टमने(System) कोणती कार्ये करावी, कोणती वर्तणूक दाखवावी, कोणते इंटरफेस(Interface) परिभाषित केले जातील आणि सिस्टमच्या(System) कार्यात कोणते अडथळे येऊ शकतात. या प्रक्रियेत विविध प्रकारची मॉडेल्स(Models) तयार केली जातात — जसे की सिनारियो-आधारित मॉडेल्स (Scenario-Based Models) जे विविध सिस्टम(System) ॲक्टर्स(Actors) च्या दृष्टीकोनातून तयार होतात, माहिती(Information) डोमेन(Domain) दाखवणारी डेटा(Data) मॉडेल्स(Models),ऑब्जेक्ट-ओरिएंटेड क्लासेस (Object Oriented Classes) आणि त्यांच्या सहकार्यांचे (Collaborations) प्रतिनिधित्व करणारी क्लास-ओरिएंटेड मॉडेल्स (Class-Oriented Models), कार्यात्मक घटकांचे (Functional Components) प्रतिनिधित्व करणारी फ्लो-ओरिएंटेड मॉडेल्स (Flow-Oriented Models) आणि बाह्य घटनांना सॉफ्टवेअरचा प्रतिसाद कसा असेल हे दाखवणारी बिहेवियरल मॉडेल्स (Behavioural Models). परिपूर्णता (Completeness), सातत्य (Consistency) आणि स्पष्टता (Clarity) यासाठी आवश्यकतांचे नीट ॲनालिसिस(Analysis) करणे गरजेचे असते. या टप्प्यावर जर काही विसंगती किंवा अस्पष्टता असेल तर ती दूर केली पाहिजे. ट्रान्सलेशन नेहमी स्पष्ट आणि संक्षिप्त स्वरूपात डॉक्यूमेंट्स(Documents) केले पाहिजे, जेणेकरून प्रकल्पात सहभागी असलेल्या प्रत्येकाला ते समजेल आणि संपूर्ण प्रकल्प जीवनचक्रात त्याचा योग्य वापर करता येईल. संपूर्ण प्रकल्पात प्रभावी ट्रान्सलेशन व्यवस्थापन करताना, आवश्यकतांमधील बदलांचा मागोवा घेतला पाहिजे, त्या बदलांची सर्व संबंधितांना जाणीव करून दिली पाहिजे, आणि ट्रान्सलेशनची पूर्तता सुरु आहे याची पडताळणी करणे आवश्यक असते.

3.1.6. डिज़ाइन ट्रांसफॉर्मेशन प्रोसेस (Design Transformation Process)

आर्किटेक्चरल डिज़ाइन(Design) प्रक्रियेत नॉन-फंक्शनल(Non-Functional) गरजा पूर्ण करण्यासाठी सॉफ्टवेअर आर्किटेक्चरचे पुन्हा पुन्हा मूल्यांकन आणि रूपांतर केले जाते. आर्किटेक्चरचे मूल्यांकन करताना सिनारियो(Scenario), सिम्युलेशन (Simulation), गणितीय मॉडेलिंग(Mathematical Modelling) आणि तर्कशक्ती तंत्र वापरले जाते. आर्किटेक्चरचे रूपांतर विविध प्रकारे केले जाऊ शकते — जसे की स्थापत्य शैली लादणे, स्थापत्य नमुने वापरणे, डिज़ाइन(Design) नमुने , वापरणे, नॉन-फंक्शनल गरजांचे कार्यक्षमता मध्ये रूपांतर करणे आणि सिस्टम(System) घटकांमध्ये त्या गरजांचे योग्य वाटप करणे. डिज़ाइन(Design) परिवर्तन प्रक्रियेदरम्यान, डेटा(Data)/क्लास(CLass) डिज़ाइन(Design), क्लास मॉडेल्स (Class Models) कडून माहिती(Information) घेऊन ती सॉफ्टवेअर अंमलबजावणीसाठी आवश्यक असलेल्या डेटा(Data) स्ट्रक्चरमध्ये(Structure) रूपांतरित केली जाते. क्लास(CLass)-जबाबदारी-सहकार्य आकृतींमध्ये परिभाषित केलेल्या वस्तू (Objects) आणि संबंध (Relationships) तसेच वर्गांची डिटेल्ड)Detailed) माहिती(Information) हे डेटा(Data) डिज़ाइन(Design) क्रियाकलापांसाठी आधार प्रदान करतात. आर्किटेक्चरल डिज़ाइन(Design) मध्ये सॉफ्टवेअरचे प्रमुख स्ट्रक्चरल (Structural) घटक (Structural Components), स्थापत्य शैली आणि डिज़ाइन(Design) नमुने यांच्यातील संबंध स्पष्ट केले जातात. तसेच, या घटकांमुळे सिस्टम(System) ट्रान्सलेशन कसे साध्य होते आणि स्थापत्य अंमलबजावणीवर कोणते अडथळे येतात हे सुद्धा यात मांडले जाते.

घटक(Component) स्तरीय डिज़ाइन(Design) मध्ये सॉफ्टवेअर आर्किटेक्चरमधील स्ट्रक्चरल (Structural) घटकांना प्रत्यक्ष सॉफ्टवेअर घटकांच्या प्रोसेसत्मक वर्णनांमध्ये रूपांतरित केले जाते. क्लास-आधारित मॉडेल्स (Class-Based Models), फ्लो मॉडेल्स (Flow Models) आणि बिहेवियरल मॉडेल्स (Behavioural Models) यामधून मिळालेली माहिती(Information) घटक डिज़ाइन(Design) क्रियाकलापांसाठी आधार म्हणून वापरली जाते. इंटरफेस(Interface) डिज़ाइन(Design) हे स्पष्ट करते की सॉफ्टवेअर अन्य सिस्टम्स (Interoperating Systems) बरोबर आणि मानव वापरकर्त्यांशी कसा संवाद करतो. यात माहितीचा(Information) फ्लो आणि विशिष्ट वर्तन प्रकार समाविष्ट असतात.

3.1.7. डेटा डिज़ाइन तत्त्वे आणि सर्वोत्तम पद्धती (Data Design Principles and Best Practices)

डेटा(Data) डिज़ाइन(Design) हे सॉफ्टवेअर अभियांत्रिकी मधील पहिल्या डिज़ाइन(Design) क्रियाकलापांचे(Activity) प्रतिनिधित्व करते. यामुळे शेवटी कमी गुंतागुंतीची, मॉड्यूलर(Modular) आणि कार्यक्षम प्रोग्राम(Program) रचना तयार होते. विश्लेषणाच्या (Analysis) टप्प्यात विकसित केलेले माहिती(Information) डोमेन(Domain) मॉडेल(Model) सॉफ्टवेअरच्या अंमलबजावणीसाठी आवश्यक असलेल्या डेटा(Data) स्ट्रक्चरमध्ये रूपांतरित केले जाते. डेटा(Data) डिक्शनरीमध्ये संग्रहित असलेली माहिती(Information) आणि एंटिटी-रिलेशनशिप डायग्राममध्ये दाखवलेले डेटा(Data) ऑब्जेक्ट्स, त्यांची वैशिष्ट्ये आणि नातेसंबंध (Relationships) हे सर्व डेटा डिज़ाइन क्रियाकलापांसाठी (Activities) मजबूत पाया प्रदान करतात.

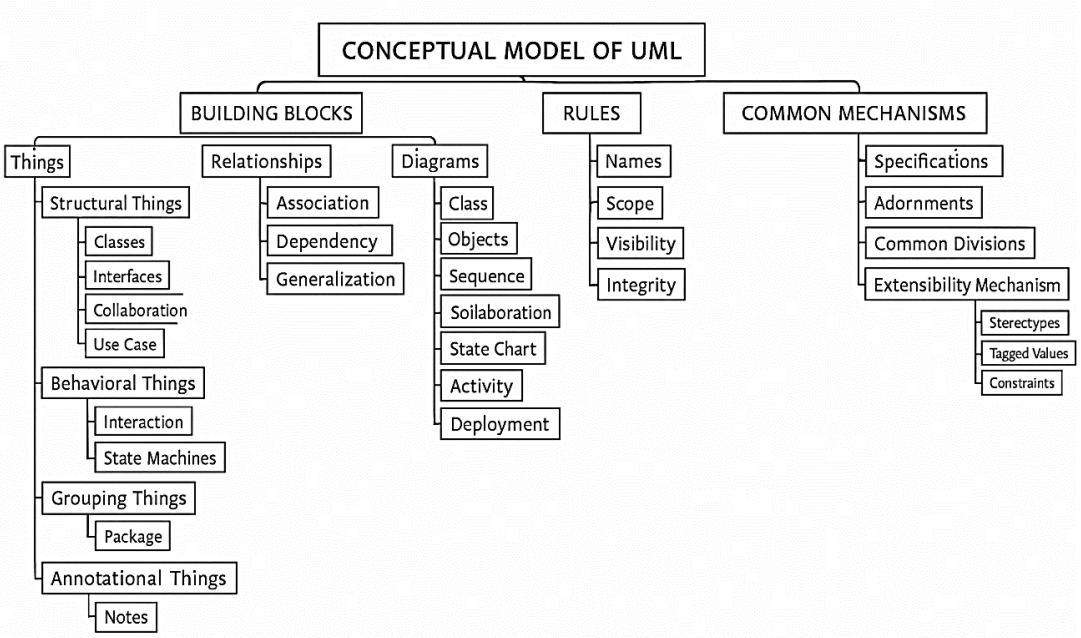


Fig. 3.5: की कॉम्पोनन्ट्स ऑफ डेटा डिज़ाइन) Key Components of Data Design)

ॲनालिसिस मॉडेल तयार करताना आणि डेटा स्ट्रक्चर डिज़ाइन करताना काही महत्त्वाची तत्त्वे लक्षात ठेवली पाहिजेत. मॉडेलने अंमलबजावणीच्या तपशीलात न अडकता, तुलनेने उच्च पातळीवरील ॲब्स्ट्रॅक्शन राखली पाहिजे आणि समस्या किंवा व्यवसाय (Business) डोमेनमध्ये दिसणाऱ्या गरजांवरच लक्ष केंद्रित केले पाहिजे. सॉफ्टवेअर अंमलबजावणीसाठी आवश्यक असलेली डेटा स्ट्रक्चर आणि त्यावर लागू होणारी ऑपरेशन्स स्पष्टपणे ओळखली पाहिजेत. वेगवेगळ्या डेटा ऑब्जेक्ट्स एकमेकांशी कसे संवाद साधतात आणि डेटा स्ट्रक्चर घटकांवर कोणते निर्बंध लादले पाहिजेत, हे दाखवण्यासाठी डेटा डिक्शनरी विकसित केली पाहिजे. डेटा(Data) डिज़ाइन(Design) प्रक्रियेत चरणवार परिष्कार (Stepwise Refinement) वापरला पाहिजे — म्हणजे डिटेल्ड)Detailed) डिज़ाइन निर्णय प्रोसेस पुढे गेल्यावरच घेतले पाहिजेत. डेटा(Data) संरचनेत संग्रहित डेटामध्ये थेट प्रवेश (Direct Access) आवश्यक असलेल्या मॉड्यूल्स ना फक्त त्या डेटा(Data) संरचनेच्या प्रतिनिधित्वाची माहिती (Information) असावी. पुनर्वापरासाठी उपयुक्त डेटा(Data) स्ट्रक्चर(Structure) आणि त्यांच्याशी संबंधित ऑपरेशन्स यांचे ग्रंथालय ठेवावे. सिस्टम)System) विकासासाठी वापरण्यात येणाऱ्या प्रोग्रामिंग भाषेने प्रभावी डेटा(Data) डिज़ाइन अंमलबजावणी सुलभ करण्यासाठी ॲब्स्ट्रॅक्शन (Abstract) डेटा(Data) प्रकारांचे समर्थन केले पाहिजे.

3.1.8. इम्प्लीमेंटेशन कंसिडरेशन्स (Implementation Considerations)

कॉन्सेप्चुअल)Conceptual) डिज़ाइन(Design) मधून लॉजिकल)Logical) डिज़ाइनमध्ये भाषांतर करताना प्रत्यक्ष डेटाबेसची अंमलबजावणी करण्यासाठी व्यावसायिक डी बी एम एस (DBMS) प्लॅटफॉर्मचा वापर केला जातो. कॉन्सेप्चुअल स्कीमा ही उच्च-स्तरीय डेटा मॉडेल्स पासून रिलेशनल (Relation) किंवा ऑब्जेक्ट-रिलेशनल डेटाबेस(Data) मॉडेल्स(Models) सारख्या अंमलबजावणी डेटा मॉडेलमध्ये रूपांतरित केली जाते. या रूपांतर प्रक्रियेला लॉजिकल डिज़ाइन किंवा डेटा(Data) मॉडेल(Model) मॅपिंग असे म्हणतात. या प्रक्रियेचा अंतिम परिणाम निवडलेल्या डीबीएमएसच्या अंमलबजावणी डेटा(Data) मॉडेलमध्ये(Model) व्यक्त झालेल्या डेटाबेस(Data) स्कीमामध्ये दिसतो.

3.1.9. फिज़िकल डिज़ाइन हा डेटाबेस(Data) डिज़ाइन(Design) प्रक्रियेचा अंतिम टप्पा आहे. या टप्प्यात डेटाबेस(Data) फायलींसाठी अंतर्गत स्टोरेज स्ट्रक्चर(Structure) , अनुक्रमणिका (Indexes), प्रवेश मार्ग (Access Paths) आणि फाइल संघटना स्पष्टपणे निर्दिष्ट (Specify) केली जाते. या टप्प्यात अनुप्रयोग कार्यक्रम , त्यांच्या उच्च-स्तरीय व्यवहार विनिर्देशांशी संबंधित डेटाबेस(Data) व्यवहार म्हणून डिज़ाइन(Design) आणि कार्यान्वित केले जातात. फिज़िकल)Physical) डिज़ाइनमध्ये(Design) सिस्टम(System) कार्यक्षमता अधिक चांगली होण्यासाठी कामगिरीची ट्रान्सलेशन, स्टोरेज मर्यादा आणि प्रवेश नमुने यांचा विचार केला जातो.

3.1.10. डेटा मॉडेलिंग सॉफ्टवेअर आणि साधने ही ट्रान्सलेशन प्रोसेस (Translation Process) सुलभ करण्यासाठी खूप महत्त्वाची भूमिका बजावतात. ही साधने आवश्यकतांचे व्हिज्युअल मॉडेलिंग शक्य करतात, त्यामुळे गैरसमज होण्याचा जोखीम कमी होते आणि भागधारकांच्या गरजा नीट समजल्या जातात याची खात्री केली जाते. ही साधने प्रारंभिक आवश्यकतांपासून ते अंतिम अंमलबजावणीपर्यंत एक स्पष्ट मार्ग देतात, त्यामुळे ट्रेसबिलिटी सुधारते आणि प्रत्येक ट्रान्सलेशन पूर्णपणे केले गेले आहे याची खात्री करता येते. व्हिज्युअल मॉडेल्स(Models) हे व्यवसाय विश्लेषक, विकसक, परीक्षक आणि भागधारक यांच्यातील संवाद सुलभ करतात. प्रभावी ट्रान्सलेशन मॉडेलिंग(Translation Modelling) मुळे आवश्यकतांची पूर्ण व्याख्या करता येते, ज्यामुळे विकास(Development) खर्च कमी होतो.

3.2 ॲनालिसिस मॉडलिंग (Analysis Modeling)

3.2.1. ॲनालिसिस मॉडेलचे घटक (Elements of Analysis Model)

ॲनालिसिस मॉडेलचे (Analysis Model) विशिष्ट घटक कोणते असतील हे वापरल्या जाणाऱ्या ॲनालिसिस)Analysis) मॉडेलिंग पद्धतीने ठरवले जाते. तरीसुद्धा, एक सामान्य घटकांचा संच असा असतो जो बहुतांश ॲनालिसिस)Analysis) मॉडेल्समध्ये (Analysis Models) समान असतो.

1. **सिनारियो-आधारित घटक (Scenario-Based element)** मध्ये सिनारियो-आधारित दृष्टिकोन (Scenario-Based Approach) वापरून वापरकर्त्याच्या दृष्टीकोनातून सिस्टम)System)चे वर्णन केले जाते. भागधारकांना (Stakeholders) या प्रक्रियेत सहभागी करणे नेहमीच एक चांगली कल्पना असते. हे करण्याचा एक चांगला मार्ग म्हणजे प्रत्येक भागधारकाने (Stakeholder) स्वतःची वापर-प्रकरणे लिहिणे, ज्यामध्ये सॉफ्टवेअर अभियांत्रिकी मॉडेल्स(Models) कसे वापरले जातील याचे स्पष्ट वर्णन केले जाते. सॉफ्टवेअर फंक्शन्ससाठी फंक्शनल-प्रोसेसिंग कथा तयार केल्या जातात, ज्या सिस्टम)System) कशी कार्य करेल हे दाखवतात. वापर-प्रकरणे म्हणजे "एक्टर)Actor)" आणि सिस्टम(System) यांच्यातील परस्परसंवादाचे वर्णन असते.
2. **क्लास-आधारित घटक (Class-Based elements)** मध्ये प्रत्येक वापर परिदृश्याचा अर्थ असा होतो की एक "ऑब्जेक्ट्स(Objects)" चा संच तयार होतो, जेव्हा एक्टर (Actor) सिस्टमशी(System) संवाद साधतो तेव्हा हे ऑब्जेक्ट्स(Objects) वापरले जातात किंवा हाताळले जातात. या वस्तूंचे वर्गांमध्ये वर्गीकरण केले जाते — म्हणजे समान गुणधर्म आणि समान वर्तन असलेल्या गोष्टींचा एकत्रित संग्रह तयार होतो. क्लासC)l(ass वेगळे करण्याचा एक सोपा मार्ग म्हणजे यूज़केस-(Use Case) लिपीमध्ये जे वर्णनात्मक संज्ञा वापरल्या जातात त्या शोधणे. त्या संज्ञांपैकी किमान काही संज्ञा उमेदवार क्लासC)l(ass म्हणून ओळखल्या जाऊ शकतात.
3. **वर्तणुकीचे घटक (Behavioural elements)** मध्ये स्टेट आकृती ही एक पद्धत आहे जिच्यामध्ये एखाद्या व्यवस्थेची अवस्था आणि त्या अवस्थेत बदल घडवून आणणाऱ्या घटना कशा असतात, याचे चित्रण केले जाते. म्हणजेच, ही आकृती सिस्टमच्या(System) वर्तनाचे दर्शन घडवते. अवस्था ही वर्तनाची कोणतीही प्रेक्षणीय पद्धत

असते — म्हणजे आपण बाहेरून पाहू शकतो की त्या क्षणी सिस्टम कोणत्या स्थितीत आहे. शिवाय, एखाद्या विशिष्ट घटनेच्या परिणामी कोणती कृती केली जाते हे देखील स्टेट आकृती मध्ये दाखवले जाते.

4. **फ्लोओरिएंटेड- घटक (Flow-Oriented Component)** मध्ये संगणकावर आधारित सिस्टम)System)तून फ्लो ऑफ इन्फॉर्मशन चे रूपांतर केले जाते. सिस्टम विविध प्रकारच्या इनपुट स्वीकारते, या इनपुटचे रूपांतर करण्यासाठी विविध कार्ये वापरली जातात, आणि शेवटी विविध स्वरूपात आउटपुट तयार केला जातो.

3.2.1.1 सिनारियो-आधारित मॉडेलिंग (Scenario-Based Modelling)

युज केस डायग्राम म्हणजे एखाद्या सिस्टमच्या घटकांमधील (System Components) परस्परसंवादाचे ग्राफिक चित्रण आहे. वापर-प्रकरणे म्हणजे केवळ सिस्टमच्या बाहेर काय आहे (म्हणजे अभिनेते (Actors)) आणि सिस्टमने काय केले पाहिजे हे स्पष्ट करण्यासाठी मदत करणारी पद्धत आहे. या संकल्पनेची समज सोपी आहे परिभाषित अभिनेत्याच्या दृष्टीकोनातून एखाद्या विशिष्ट वापराच्या परिस्थितीचे सोप्या भाषेत वर्णन करायचे.

1. सीमा (System Boundary) — जी सिस्टम)System)ची सभोवतालच्या जगाशी असलेली मर्यादा किंवा परिभाषा ठरवते.
2. अभिनेते (Actors) — सामान्यत: सिस्टम)System)शी संबंधित व्यक्ती किंवा डिव्हाइस (Devices), जी त्यांच्या भूमिकेनुसार परिभाषित केली जातात.
3. यूज़-प्रकरणे (Use Cases) — जी विशिष्ट भूमिका सिस्टम)System)मध्ये किंवा तिच्या आसपास पार पाडली जाते हे दर्शवतात.
4. अभिनेत्यांमधील आणि यूज़-प्रकरणांमधील संबंध
5. "वापराचा धागा (Usage Thread)" — एखाद्या सिस्टम)System)साठी वापराचा फ्लो किंवा परिस्थिती कसे असेल, हे वर्णन करणारी माहिती(Information).
5. अभिनेते (Actors) — सिस्टम(System) फंक्शन म्हणून लोक किंवा डिव्हाइस बजावलेल्या भूमिकांचे प्रतिनिधित्व करतात.
6. वापरकर्ते (Users) — दिलेल्या परिस्थितीनुसार अनेक वेगवेगळ्या भूमिका बजावू शकतात.

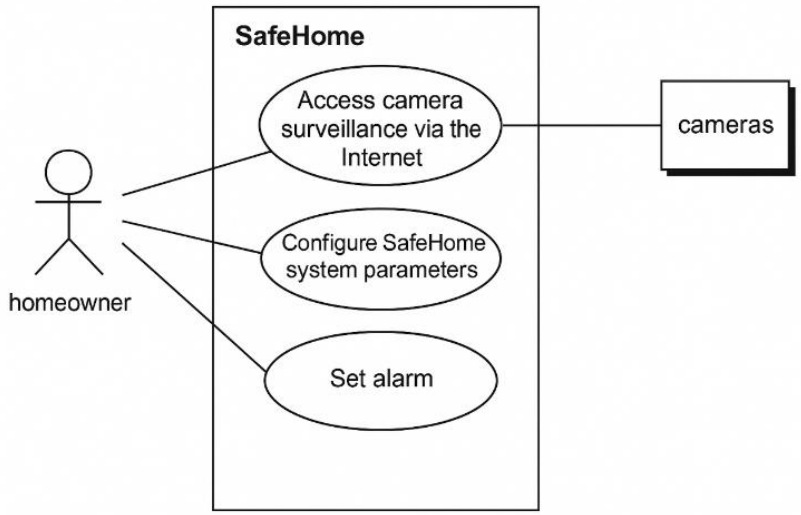


Fig. 3.6 : उदाहरण 1 : युज केस डायग्राम फॉर सिस्टम मॅनेजमेंट (Example 1: use case diagram for hotel management system)

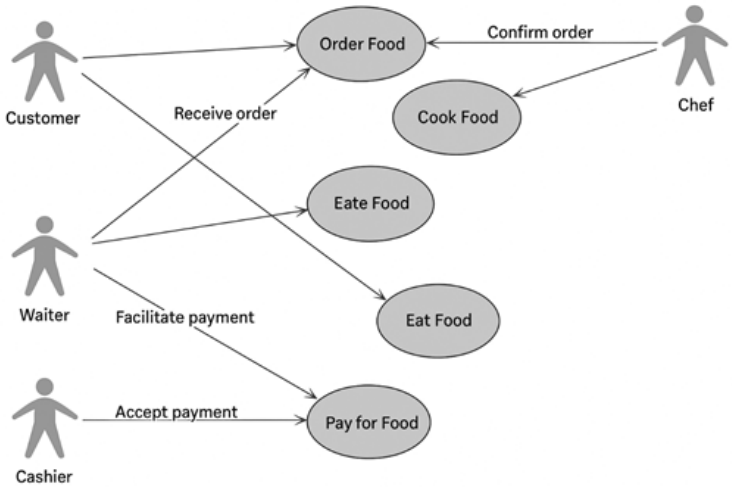


Fig. 3.7: उदाहरण 2 : युज केस डायग्राम फॉर ए टी एम सिस्टम (Example 2: use case diagram for ATM system)

3.2.1.2 क्लास बेस्ड मॉडेलिंग (Class Based Modelling)

क्लास :

एका क्लास चे प्रतिनिधित्व आयताकृती (Rectangle) ने केले जाते, ज्यामध्ये तीन विभाग असतात:

- सर्वात वरचा विभाग — क्लास चे नाव दाखवतो.
- मधला विभाग — क्लास चे ॲट्रीब्यूट्स दाखवतो.
- खालचा विभाग — क्लास च्या ऑपरेशन्सचे प्रतिनिधित्व करतो.

ॲट्रीब्यूट्स (Attributes) आणि ऑपरेशन्स (Operations) ची व्हिजिबिलिटी (Visibility) खालीलप्रमाणे दर्शवली जाते:

- **पब्लिक (Public)** : पब्लिक सदस्य संपूर्ण सिस्टम(System) मध्ये कोठूनही दिसतो. क्लास डायग्राम **(Class Diagram)** मध्ये याला **'+'** या चिन्हाने दर्शवले जाते.
- **प्रायव्हेट (Private) :** प्रायव्हेट सदस्य (Private Member) केवळ क्लासच्या आतूनच (Within the Class) दिसतो; क्लासच्या बाहेरून प्रवेश करता येत नाही. प्रायव्हेट सदस्याला(Private Member) **'–'** या चिन्हाने दर्शवले जाते.
- **प्रोटेक्टेड (Protected)** :प्रोटेक्टेड सदस्य (Protected Member) क्लासच्या आतून आणि त्या क्लासच्या सब-क्लासेसमधून (Subclasses) दिसतो, पण बाहेरून दिसत नाही. प्रोटेक्टेड सदस्याला (Protected Member) **'#'** या चिन्हाने दर्शवले जाते.

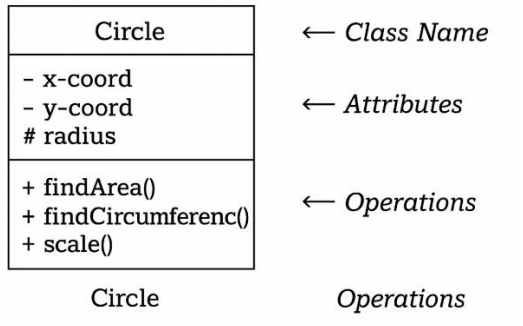


Fig. 3.8: क्लास (Class)

ऑब्जेक्ट ला आयताकृती (Rectangle) द्वारे दोन विभागांसह दर्शविले जाते.

वरच्या विभागात त्या क्लास किंवा पॅकेजच्या नावासह ऑब्जेक्टचे नाव दाखवले जाते, ज्याचे ते उदाहरण (Instance) असते. हे नाव खालीलपैकी कोणत्याही स्वरूपात असते:

ऑब्जेक्ट-नाव (Object Name) : क्लास-नाव(Class Name)

ऑब्जेक्ट-नाव(Objec Name) : क्लास-नाव(Class Name) :: पॅकेज-नेम(Package Name)

क्लास-नाव(Class Name) — जर ती निनावी ऑब्जेक्ट (Anonymous Object) असेल.
खालच्या विभागात ॲट्रीब्यूट्स (Attributes) च्या मूल्यांचे (Values) प्रतिनिधित्व केले जाते. हे खालील स्वरूपात लिहिले जाते:
ॲट्रीब्यूट-नाव(Object Name) = मूल्य(Value)
कधीकधी गोलाकार आयताकृती (Rounded Rectangle) वापरून सुद्धा ऑब्जेक्टचे प्रतिनिधित्व (Object Representation) केले जाते.
उदाहरण : आपण क्लास(Class) वर्तुळ (Circle) मधील C1 नावाच्या ऑब्जेक्ट(Object) चा विचार करूया. गृहीत धरूया की C1 चे सेंटर (Centre) = (2, 3) आहे आणि C1 ची त्रिज्या (Radius) = 5 आहे. खालील आकृती (Diagram) या ऑब्जेक्टचे (Object) चित्रण करते.

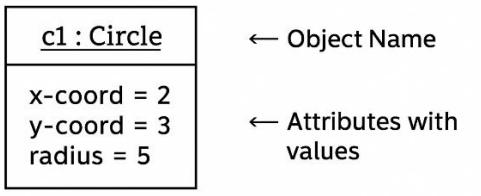


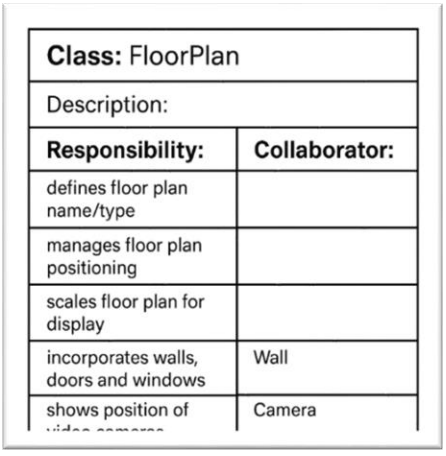
Fig. 3.9: ऑब्जेक्ट (Object)

3.2.1.3. क्लास रिस्पॉन्सिबिलिटी कोलॅबोरेटर (सी आर सी) मॉडेलिंग(Class Responsibility Collaborator Modelling)

क्लास रिस्पॉन्सिबिलिटी कोलॅबोरेटर (सीआरसी) मॉडेलिंग(Class Responsibility Collaborator Modelling) ही सिस्टम(System) किंवा उत्पादनाच्या आवश्यकतांशी (Product Requirements) संबंधित क्लासेस (Classes) ओळखण्यासाठी आणि संघटित (Organize) करण्यासाठी एक सोपी पद्धत (Simple Technique) आहे.

सी आर सी मॉडेलिंग खालीलप्रमाणे समजावले जाते :

सी आर सी मॉडेल म्हणजे खरं तर स्टँडर्ड इंडेक्स कार्ड्स (Student Index Card) चा एक संग्रह आहे जो क्लासेस (Classes) चे प्रतिनिधित्व करतो. हे कार्ड तीन विभागांमध्ये विभागलेले असते. कार्डच्या वरच्या भागात तुम्ही क्लास(Class)चे नाव लिहिता कार्डच्या मुख्य भागात (Body) डाव्या बाजूला क्लास(Class)च्या रिस्पॉन्सिबिलिटीज(Responsibilities) आणि उजव्या बाजूला कोलॅबोरेटर(collaborator)ची यादी करता."
रिस्पॉन्सिबिलिटीज(Responsibilities) म्हणजे त्या क्लास(Class)साठी समर्पक अशी वैशिष्ट्ये आणि फंक्शन्स(Functions) म्हणजेच "क्लासला काय माहित आहे किंवा काय करते". कोलॅबोरेटर(collaborator) म्हणजे असे क्लासेस(Classes) किंवा घटक(Components) जे त्या क्लास(Class)ला त्याच्या रिस्पॉन्सिबिलिटीज (Responsibilities) पूर्ण करण्यासाठी आवश्यक माहिती)Required (Information किंवा एक्शन(Action) पुरवतात. साधारणपणे, कोलॅबोरेशन म्हणजे एकतर माहितीची(Information) विनंती किंवा एखाद्या कृतीची विनंती असते.



Class: FloorPlan	
Description:	
Responsibility:	**Collaborator:**
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of	Camera

Fig. 3.10. उदाहरण 1: क्लास डायग्राम फॉर रेल्वे मॅनेजमेंट सिस्टम (Example 1: class diagram for railway management system)

3.2.1.4 वर्तणुकीचे घटक (Behavioral Elements)

बिहेवियरल घटक (Behavioral Elements) हे वर्णन करतात की सिस्टम(System) गतिशीलपणे (Dynamically) कशी वागते म्हणजे ती बाह्य (External) आणि अंतर्गत (Internal) घटनांना (Events) कसा प्रतिक्रिया (Respond) देते.
हे घटक सिस्टम(System) ऑपरेशन दरम्यान होणाऱ्या क्रिया (Actions), स्थिती (States) आणि परस्परसंवादच्या अनुक्रमाचे मॉडेल(Model) तयार करतात.

उद्देश (Objectives) :

- वस्तू आणि सिस्टमच्या(System) गतिशील वर्तनाचे मॉडेल(Model) तयार करणे.
- स्टेट चेंजेस(State Changes) आणि इव्हेंट-आधारित(Event Based) वर्तन समजून घेणे.
- प्रक्रियेदरम्यान वस्तूंमधील परस्परसंवाद दाखवणे.

कॉमन बिहेविरल मॉडेल (Common Behevioural Model)

मॉडेल(Model)	वर्णन
स्टेट ट्रान्झिशन डायग्राम	स्टेट ट्रान्झिशन डायग्राम हे एखाद्या वस्तूचे किंवा सिस्टमचे(System) वर्तन दर्शवते, जिथे घटनांच्या आधारे ती एका अवस्थेतून दुसऱ्या अवस्थेत कशी जाते हे दाखवले जाते.
सिक्वेन्स डायग्राम	सिक्वेन्स डायग्राम हे कालक्रमानुसार वस्तूंमधील परस्परसंवादाचे दृश्यरूप सादर करतो.
ॲक्टिव्हिटी डायग्राम	ॲक्टिव्हिटी डायग्राम प्रोसेस(Process) किंवा कार्यप्रवाहातील क्रियाकलापांच्या प्रवाहाचे प्रतिनिधित्व करतो.

3.2.1.5. फ्लो-ओरिएंटेड घटक (Flow Oriented Components)

फ्लो-ओरिएंटेड घटक हे सिस्टम(System)द्वारे डेटा(Data) कसा चालतो आणि त्यावर प्रोसेस(Process) कशी केली जाते याचे वर्णन करतात. हे घटक डेटा(Data) फ्लो , प्रोसेस(Process) आणि सिस्टम(System)मध्ये होणाऱ्या परिवर्तनांवर लक्ष केंद्रित करतात.

उद्देश:

- सिस्टममधील(System) डेटाच्या(Data) प्रवाहाचे मॉडेल(Model) तयार करणे
- प्रोसेस(Process) आणि डेटासह(Data) त्यांचे परस्परसंवाद दर्शविणे
- इनपुट(Input) → प्रोसेस(Process) → आउटपुट संबंध दर्शविणे

कॉमन डेटा फ्लो मॉडेल (Common Data Flow Model):

मॉडेल(Model)	वर्णन
डेटा(Data) फ्लो डायग्राम (डीएफडी)	सिस्टममधून(System) डेटा(Data) कसा प्रवाहित होतो आणि त्यावर प्रोसेस(Process) कशी केली जाते हे दर्शविते.
डेटा(Data) मॉडेल(Model) (ईआर आकृती)	डेटा(Data) संस्था आणि त्यांच्या संबंधांचे प्रतिनिधित्व करते.

3.3. डिज़ाइन मॉडेलिंग(Design Modelling)

3.3.1. फनडामेंटल डिझाईन कंसेप्ट्स(Fundamental Design Concepts)

प्रत्येक बौद्धिक रचना ही मूलभूत संकल्पना आणि विशिष्ट तंत्रांद्वारे वैशिष्ट्यीकृत असते. विशिष्ट परिस्थितीत ही तंत्रे लागू केली जातात. तंत्रज्ञानातील बदल, बौद्धिक फॅड, आर्थिक परिस्थिती आणि सामाजिक चिंता यांच्या प्रभावामुळे ही तंत्रे येतात आणि कालांतराने बदलतात किंवा कालबाह्य होतात. सॉफ्टवेअर डिज़ाइन)Software (Design च्या मूलभूत संकल्पना म्हणजेच: ॲब्स्ट्रॅक्शन(Abstraction), रचना (Structure), माहिती(Information) लपविणे, मॉड्युलरिटी(Modularity),कॉनकरन्सी(Concurrency), पडताळणी आणि डिज़ाइन(Design) सौंदर्यशास्त्र (Design Aesthetics) — यांचा समावेश होतो.

3.3.1.1. ॲब्स्ट्रॅक्शन(Abstraction)

ॲब्स्ट्रॅक्शन(Abstraction) हे एक बौद्धिक साधन आहे जे आपल्याला त्या संकल्पनांच्या विशिष्ट उदाहरणांपासून वेगळं राहून, संपूर्ण संकल्पना हाताळण्याची मुभा देते. ट्रान्झलेशन(Translation) परिभाषा आणि डिज़ाइन(Design) दरम्यान, ॲब्स्ट्रॅक्शन(Abstraction) आपल्याला अंमलबजावणीच्या तपशीलांपासून सिस्टमच्या(System) वैचारिक पैलूंना वेगळे करण्यास मदत करते.

उदाहरणार्थ, स्टॅक(Stack) आणि क्यू **(**queue**)** लागू करण्यासाठी नेमकी प्रतिनिधित्व योजना कशी आहे याची चिंता न करता, क्यू(Queue)ची एफआयएफओ (FIFO) किंवा स्टॅकची एलआयएफओ (LIFO) मालमत्ता निर्दिष्ट करता येते. त्याचप्रमाणे, आपण दिनचर्येच्या अल्गोरिदम तपशीलांपासून स्वतंत्रपणे, डेटा(Data) स्ट्रक्चर(Structure) **(**उदा. न्यू(New), पुश**(**Push**)**, पॉप(Pop**)**, टॉप**(**Top**)**, एमटी(Empty) हाताळणाऱ्या दिनचर्यांची कार्यक्षमता निश्चित करू शकतो.

ॲब्स्ट्रॅक्शन(Abstraction) मुळे गुंतागुंतीचे प्रमाण कमी होते. सॉफ्टवेअर डिज़ाइन)Software मध्ये(Design सर्वसाधारणपणे तीन प्रकारच्या ॲब्स्ट्रॅक्शन(Abstraction) यंत्रणा वापरल्या जातात फकशनल ॲब्स्ट्रॅक्शन**(**Functional Abstraction), डेटा ॲब्स्ट्रॅक्शनData) Abstraction(, कंट्रोल ॲब्स्ट्रॅक्शन(Control Abstraction)

फकशनल ॲब्स्ट्रॅक्शन(Functional Abstraction) मध्ये पॅरामीटराइज्ड सबप्रोग्राम्सचा वापर केला जातो. उपप्रोग्राम)**Sub** मध्ये(Program पॅरामीटर देण्याची क्षमता आणि वेगवेगळ्या कॉल्समध्ये वेगवेगळ्या पॅरामीटर मूल्यांचे बंधन घालण्याची क्षमता ही एक शक्तिशाली ॲब्स्ट्रॅक्शन यंत्रणा आहे.

ही ॲब्स्ट्रॅक्शन उपकार्यक्रमांच्या संग्रहांमध्ये सामान्यीकृत केली जाते, ज्याला "गट" **(**उदा. एडा**(**ada**)** मधील पॅकेज, किंवा क्लस्टर्स**)** म्हणतात.

डेटा(Data) ॲब्स्ट्रॅक्शन(Abstraction) मध्ये, वस्तूंवरील कायदेशीर कार्ये निर्दिष्ट करून डेटा(Data) प्रकार किंवा डेटा(Data) ऑब्जेक्ट(Object) ठरवले जातात; प्रतिनिधित्व आणि हेरफेर करण्याचे तपशील दडपले जातात. "डेटा एन्कॅप्सुलेशन Data) Encapsulation(" हा शब्द त्या डेटा (Data) ऑब्जेक्टच्या उदाहरणाला दर्शवतो, जिथे डेटा (Data) ऑब्जेक्टवर केले जाणारे ऑपरेशन्स निश्चित असतात. "ॲब्स्ट्रॅक्ट डेटा टाईप) Abstract Data Type(" हा शब्द त्या डेटा(Data) प्रकाराच्या घोषणेसाठी वापरला जातो **(**उदा. स्टॅक**)** जिथून अनेक उदाहरणे तयार केली जाऊ शकतात.

कंट्रोल ॲब्स्ट्रॅक्शनचा वापर असा केला जातो की नेमकं कंट्रोल कसं करायचं हे न सांगता, इच्छित परिणाम कसा साधायचा हे ठरवता येते. उदा. आधुनिक प्रोग्रामिंग भाषांतील स्टेटमेंट्स आणि विधाने ही मशीन कोड अंमलबजावणीची कंट्रोल ॲब्स्ट्रॅक्शन(Control Abstraction) असतात, जिथे सशर्त उडी सूचना वापरल्या जातात.

3.3.1.2 इन्फॉर्मेशन लपविणे (Information Hiding)

इन्फॉर्मेशन लपविणे ही सॉफ्टवेअर साठी एक मूलभूत डिज़ाइन(Design) संकल्पना आहे.ही संकल्पना पर्णस यांनी मांडली होती. जेव्हा एखादी सॉफ्टवेअर सिस्टम**)**Software System) इन्फॉर्मेशन(Information) लपविण्याच्या दृष्टिकोनाचा वापर करून डिज़ाइन(Design) केली जाते, तेव्हा सिस्टममधील(System) प्रत्येक मॉड्यूल आपल्या प्रोसेस(Process) क्रियाकलापांचे अंतर्गत इन्फॉर्मेशन(Internal Information) लपवते आणि प्रत्येक मॉड्यूल(Module) फक्त चांगल्या प्रकारे परिभाषित इंटरफेस(Interface) द्वारेच संवाद साधते. फंक्शनल ॲब्स्ट्रॅक्शन(Functional Abstraction), डेटा ॲब्स्ट्रॅक्शन(Data Abstraction) आणि कंट्रोल ॲब्स्ट्रॅक्शन(Control Abstraction) या सर्वांमध्ये इन्फॉर्मेशन(Information) लपविण्याची वैशिष्ट्ये दिसून येतात. पर्णस यांच्या मते, डिज़ाइनची(Design) सुरुवात नेहमी कठीण डिज़ाइन निर्णय(Difficult Design Decision) आणि अशा डिज़ाइन(Design) निर्णयांच्या यादीपासून करावी जी बदलण्याची शक्यता असते. प्रत्येक मॉड्यूल(Module) असे डिज़ाइन(Design) करायचे की ते इतर मॉड्यूल्सपासून तो निर्णय लपवेल. कठीण आणि बदलण्यायोग्य डिज़ाइन(Design) निर्णय लपविण्याव्यतिरिक्त, इन्फॉर्मेशन लपविण्यासाठी इतर उमेदवारांमध्ये खालील गोष्टी समाविष्ट होतात:

1. डेटा(Data) रचना, त्याची अंतर्गत जोडणी आणि त्यात फेरफार करणाऱ्या कार्यपद्धतींचे अंमलबजावणी तपशील **(**यालाच डेटा(Data) ॲब्स्ट्रॅक्शन चे तत्त्व म्हणतात**)**.
2. ऑपरेटिंग सिस्टम(System) मधील रांगेसारख्या कंट्रोल ब्लॉक्सचे स्वरूप.
3. कॅरेक्टर कोड, कॅरेक्टर सेटची ऑर्डर आणि इतर अंमलबजावणी तपशील.
4. शिफ्टिंग, मास्किंग आणि मशीनवर अवलंबून असलेले इतर तपशील.

इन्फॉर्मेशन लपविणे एखाद्या सिस्टम) **System)**च्या स्थापत्य डिज़ाइनसाठी(Design) एक मुख्य डिज़ाइन(Design) तंत्र म्हणून किंवा मॉड्युलायझेशन निकष म्हणून वापरले जाऊ शकते.

3.3.1.3 स्ट्रक्चर (Structure)

स्ट्रक्चर(Structure) हे संगणक सिस्टम)System)चे मूलभूत वैशिष्ट्य आहे. हे मोठ्या सिस्टम)Big System)चे लहान(Small), अधिक व्यवस्थापित करण्यायोग्य एककांमध्ये विघटन करण्यास अनुमती देते ज्यात सिस्टममधील(System) इतर एककांशी चांगले परिभाषित संबंध असतात. सिस्टम(System) रचनेचा सर्वात सामान्य प्रकार म्हणजे नेटवर्क.

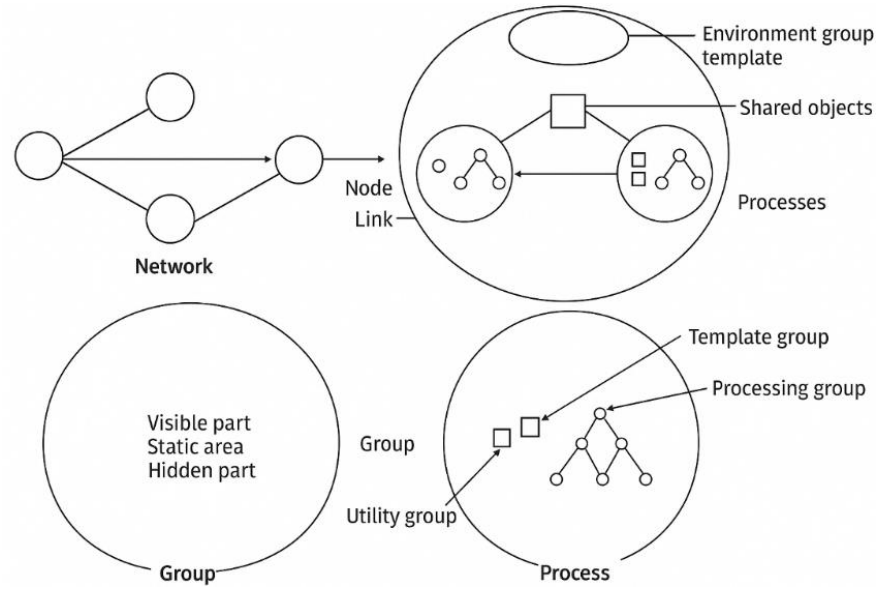


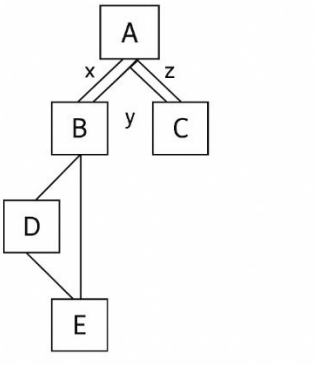
Fig. 3.11: सॉफ्टवेअर सिस्टम स्ट्रक्चर) Software System (Structure

एक संगणकीय नेटवर्क निर्देशित आलेख म्हणून दर्शविले जाऊ शकते, ज्यामध्ये नोड्स(Nodes) आणि आर्क्स(Arcs) असतात. नोड्स(Nodes) हे प्रोसेस(Process) घटकांचे प्रतिनिधित्व करतात जे डेटा(Data) रूपांतरित करतात आणि आर्क्स(Arcs) चा वापर नोड्समधील डेटा(Data) दुवे दर्शविण्यासाठी केला जातो. नोड्स डेटा स्टोअर)Nodes Data Storeचे(आणि आर्क्स डेटा)Arcs (Data परिवर्तनाचे प्रतिनिधित्व करू शकतात.

एक नेटवर्क एका उपप्रोग्राम)Sub मध्ये(Program डेटा फ्लो(Data Flow) आणि प्रोसेस(Process) चरण किंवा अनुक्रमिक उपप्रोग्रामच्या(Program) संग्रहातील डेटा फ्लो(Data Flow)कसा आहे हे निर्दिष्ट करते.संगणकीय नेटवर्कचा सर्वात गुंतागुंतीचा प्रकार म्हणजे वितरित संगणकीय सिस्टम)System**)** जिथे प्रत्येक नोड**(**node**)** हा खाजगी मेमरी असलेल्या भौगोलिकदृष्ट्या वेगळ्या प्रोसेसरचे प्रतिनिधित्व करतो.

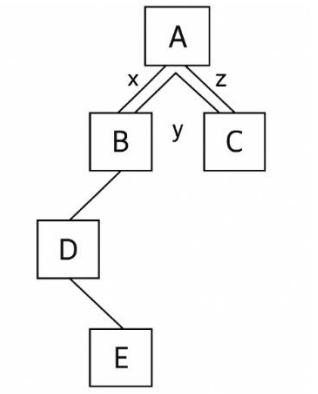
गुंतागुंतीच्या प्रोसेसिंग नोडच्या आतील संरचनेत समवर्ती प्रोसेस **(**कॉनकरंट प्रोसेसिंग**)** असतात ज्या समांतरपणे **(**पॅरलल**)** कार्यान्वित होतात आणि सामायिक चर (Shared Variables) व समकालिक **(**सिन्क्रोनस**)** व अतुल्यकालिक **(**असिन्क्रोनस**)** संदेश पासिंगच्या काही संयोजनाद्वारे संवाद साधतात. "उपयोग" आणि पूरक संबंध "वापरले जातात" हे संबंध सॉफ्टवेअर सिस्टम)Software मधील(System ऍब्स्ट्रॅक्शन**(**Abstraction**)** च्या श्रेणीबद्ध क्रमासाठी **(**Hierarchy of Abstraction**)** एक आधार प्रदान करतात. "उपयोग" संबंध निर्देशित आलेख (Directed Graph) म्हणून दर्शविले जाऊ शकतो, जिथे ए → बी चा अर्थ "ए बी वापरतो" किंवा "बी ए द्वारे वापरला जातो".

ऍब्स्ट्रॅक्शन चा श्रेणीबद्ध क्रम खालील नियमाने स्थापित केला जातो: जर अ आणि ब या भिन्न संस्था असतील, आणि जर अ ब वापरत असेल, तर ब ला अ किंवा अ वापरणारी कोणतीही संस्था वापरण्याची परवानगी नाही.यामुळे ऍब्स्ट्रॅक्शन मधील स्पष्ट आणि एकमार्गी अवलंबित्व (Unidirectional Dependency) राखले जाते.



A graph structure chart

Fig. 3.12 : ग्राफ स्ट्रक्चर चार्ट (Graph Structure Chart)



A tree structure chart

Fig. 3.13 : ट्री स्ट्रक्चर चार्ट टाइप (Tree Structure Chart)

झाडामध्ये (Tree) मुळापासून प्रत्येक नोड(Node) पर्यंत एक अनोखा मार्ग (Unique Path) असतो. अचक्रीय, निर्देशित आलेखामध्ये (Acyclic Directed Graph) मुळापासून नोड पर्यंत एकापेक्षा जास्त मार्ग असू शकतात. अशा प्रकारच्या आकृतींना स्ट्रक्चर(Structure) चार्ट म्हणतात.

3.3.1.4. मॉड्युलरिटी (Modularity)

मॉड्युलरिटी ही चिंतांच्या पृथक्करणाची (Separation of Concerns) सर्वात सामान्य अभिव्यक्ती आहे. सॉफ्टवेअर स्वतंत्रपणे नामांकित आणि संबोधित करण्यायोग्य घटकांमध्ये विभागले जाते, ज्यांना कधीकधी मॉड्यूल(Module) असे म्हणतात, आणि हे मॉड्यूल्स(Modules) एकत्रित करून समस्येच्या गरजा पूर्ण करण्यासाठी सिस्टम(System) तयार केली जाते. असे म्हटले गेले आहे की "मॉड्युलरिटी(Modulrity) हे सॉफ्टवेअरचे एकमेव वैशिष्ट्य आहे जे प्रोग्रामला(Program) बौद्धिकरित्या व्यवस्थापित करण्यास अनुमती देते."

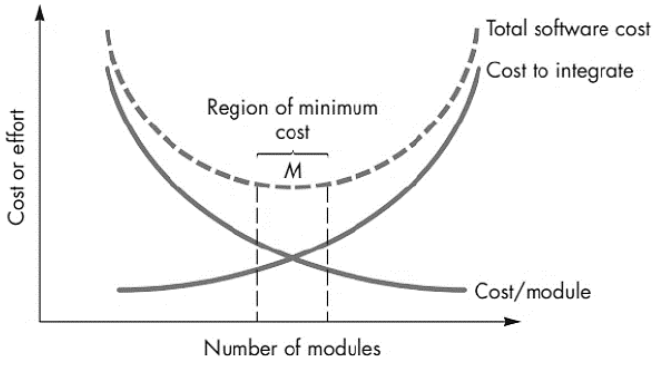


Fig. 3.14 : मॉड्युलरिटी (Modularity)

Fig. 3.3.1.4च्या संदर्भाने पाहिल्यास, मॉड्यूलची एकूण संख्या वाढत असताना वैयक्तिक सॉफ्टवेअर मॉड्यूल(Module) विकसित करण्यासाठी लागणारा प्रयत्न (खर्च) कमी होतो.समान आवश्यकतांच्या संच पाहता, अधिक मॉड्यूल्स(Modules) असल्याने प्रत्येक मॉड्यूलचा आकार लहान असतो.तथापि, मॉड्यूल्सची संख्या जसजशी वाढते,

तसतसे मॉड्यूल्स(Modules) एकत्रित करण्याशी (Integration) संबंधित प्रयत्न (खर्च) मात्र वाढतो. या वैशिष्ट्यांमुळे आकृतीमध्ये दाखवलेला एकूण खर्च किंवा प्रयत्नांचा वक्र (Overall Effort Curve) तयार होतो.
एक अशी मॉड्यूल्सची संख्या, M, असते ज्यामुळे कमीतकमी विकास खर्च (Minimum Development Cost) साधता येतो. आकृतीमध्ये दाखवलेला वक्र मॉड्युलरिटी(Modularity) चा विचार करताना उपयुक्त गुणात्मक मार्गदर्शन (Qualitative Guidance) पुरवतो. आपण मॉड्युलाइझ केले पाहिजे, पण M च्या आसपास राहण्याची काळजी घ्यायला हवी. अंडर-मॉड्युलरिटी किंवा ओव्हर-मॉड्युलरिटी टाळली पाहिजे.

3.3.1.5. कॉनकरन्सी (Concurrency)

सॉफ्टवेअर सिस्टिम्स(Software Systems) या सीक्वेंशियल किंवा कॉनकरंट म्हणून वर्गीकृत केल्या जाऊ शकतात. सीक्वेंशियल सिस्टिममध्ये, सिस्टिमचा केवळ एक भाग कोणत्याही वेळेस सक्रिय असतो. कॉनकरंट सिस्टिममध्ये, स्वतंत्र प्रोसेस(Process) असतात ज्या अनेक प्रोसेसर उपलब्ध असल्यास एकाच वेळी सक्रिय होऊ शकतात. एकाच प्रोसेसरवर, कॉनकरंट प्रोसेस(Process) अंमलबजावणीच्या वेळेत परस्परपूरक (Interleaved) पद्धतीने चालवल्या जाऊ शकतात. यामुळे टाइम-शेअरिंग(Time Sharing), मल्टी-प्रोग्रामिंग(Multi Programming) आणि रिअल-टाइम सिस्टिम्स(Real Time Systems) च्या अंमलबजावणीस परवानगी मिळते.
कॉनकरंट सिस्टिम मध्ये काही अद्वितीय समस्या उद्भवतात, जसे:

1. **गतिरोध (Deadlock):** ही एक अवांछित परिस्थिती आहे, जी तेव्हा घडते जेव्हा सिस्टिममधील सर्व प्रोसेस(Process) इतर प्रक्रियांकडून काही क्रिया पूर्ण होण्याची वाट पाहत राहतात आणि त्यामुळे कोणतीही प्रोसेस(Process) पुढे जाऊ शकत नाही.
2. **परस्पर बहिष्कार (Mutual Exclusion):** हे अत्यावश्यक असते कारण अनेक प्रोसेस(Process) एकाच वेळी सामायिक प्रोसेसिंग स्टेट मधील समान घटक अद्ययावत करण्याचा प्रयत्न करू नयेत, याची खात्री करण्यासाठी परस्पर बहिष्कार आवश्यक आहे.
3. **सिंक्रोनाइझेशन (Synchronization):** सिंक्रोनाइझेश(Synchronization) आवश्यक आहे जेणेकरून वेगवेगळ्या अंमलबजावणी वेगाने कार्य करणाऱ्या कॉनकरंट प्रोसेस(Process), त्यांच्या अंमलबजावणी इतिहासातील योग्य बिंदूंवर संवाद साधू शकतील.

3.3.1.6.1व्हेरीफिकेशन (Verification)

व्हेरिफिकेशन(verification) ही सॉफ्टवेअर डिज़ाइन)Software मधील एक मूलभूत संकल्पना आहे(Design. डिज़ाइनम्हणजे ग्राहकांच्या गरजा आणि त्या गरजा पूर्ण करणारी अंमलबजावणी यांच्यातील सेतू असतो (Design). डिज़ाइनमुळे ग्राहकांच्या गरजा पूर्ण करणारी अंमलबजावणी होईल (Design), हे दाखवले गेले की डिज़ाइनची (Design) व्हेरिफिकेशन केली जाते.
ही प्रोसेस सामान्यतः दोन चरणांमध्ये केली जाते:

1. सॉफ्टवेअर ट्रान्सलेशन ची व्याख्या ग्राहकांच्या गरजा पूर्ण करते की नाही याची व्हेरिफिकेशन — याला आवश्यकतांची व्हेरिफिकेशन (Requirements Verification) म्हणतात.
2. डिज़ाइन व्याख्या ही आवश्यकता पूर्ण करते की नाही याची व्हेरिफिकेशन — याला डिज़ाइनची (Design) व्हेरिफिकेशन(Design Verification) म्हणतात.

3.3.1.7 एस्थेटिक्स (Aesthetics)

कला असो वा तंत्रज्ञान, डिज़ाइनसाठी(Design) एस्थेटिक्स विचार मूलभूत असतात.
साधेपणा, लालित्य आणि उद्देशाची स्पष्टता हीच ती वैशिष्ट्ये आहेत जी उत्कृष्ट गुणवत्तेची उत्पादने मध्यम दर्जाच्या उत्पादनांपासून वेगळी करतात.
कला असो वा तंत्रज्ञान, डिज़ाइनसाठी(Design) एस्थेटिक्स हे मूलभूत आहे.
यामध्ये पुढील बाबींचा समावेश होतो:

- साधेपणा (Simplicity)
- स्पष्टता (Clarity)

- लालित्य (Elegance)
- उत्पादनाची गुणवत्ता (Product Quality)

3.4 डिज़ाइन नोटेशन(Design Notation)

हा विभाग तांत्रिक वैशिष्ट्यांमध्ये सिस्टम(System) आवश्यकतांचे ट्रान्सलेशन(Transaltion) करण्यासाठी महत्त्वपूर्ण असलेल्या तीन मूलभूत डिज़ाइन(Design) नोटेशन्सचा शोध घेतो.

ही साधने एमएसबीटीई(MSBTE) अभ्यासक्रमात विशेषतः जोर दिलेल्या संरचित ॲनालिसिस)Analysis) आणि डिज़ाइन पद्धतींचा कणा आहेत.

3.4.1. डेटा फ्लो डायग्राम (डीएफडी) नोटेशन (Data flow diagram-Notations)

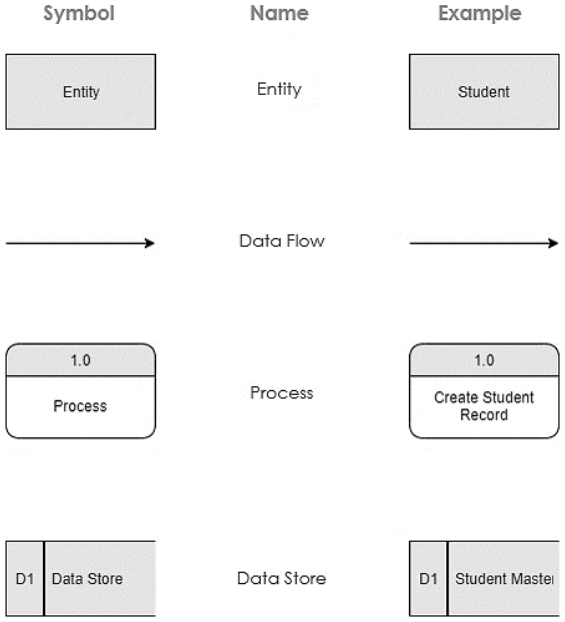


Fig. 3.15: डेटा फ्लो डायग्राम (डी एफ डी) नोटेशन (Data Flow Diagram (DFD) Notations)

.3.4.2 कॉन्सेप्चुअल फ्रेमवर्क(Conceptual Framework)

डेटा फ्लो डायग्राम(Data Flow Diagram-डीएफडी) हे इनपुट(Input), प्रोसेस(Process), डेटा स्टोरेज(Data Storage) आणि आउटपुट(Output) ओळखून सिस्टमद्वारे(System) इन्फॉर्मेशनच्या प्रवाहाचे दृश्यमान प्रतिनिधित्व करतात. ते उच्च-स्तरीय ट्रान्सलेशन(Transaltion) आणि डिटेल्ड)Detailed) तांत्रिक डिज़ाइन(Technical Design) दरम्यान एक पूल(Bridge) म्हणून कार्य करतात, ज्यामुळे विद्यमान प्रणालींचे ॲनालिसिस)Analysis) करण्यासाठी किंवा नवीन डिज़ाइन(Design) करण्यासाठी ते अपरिहार्य ठरतात.

3.4.3. मुख्य घटक (Main Components)

1. **प्रोसेस(Process)** : प्रोसेस(Process) म्हणजे क्रियापद वाक्यांशांसह लेबल केलेल्या डेटाच्या(Data) रूपांतरांचे प्रतिनिधित्व करणे **(**उदा**.,** "व्हॅलिडेट पेमेंट"**).** गॅन-सरसन नोटेशन मध्ये**,** प्रोसेस(Process) गोलाकार आयत (Rounded Rectangle) वापरून दर्शवली जाते**,** तर योर्डन **/** डीमार्को नोटेशन मध्ये वर्तुळे (Circle) वापरतात**.**

   नोटेशन (Notation):

   - गोलाकार आयत ही एक प्रोसेस दर्शवते**.**
   - प्रक्रियांना ओळखण्यासाठी आयडी (ID) दिले जातात हे सोप्या संदर्भासाठी वापरले जाते**.**

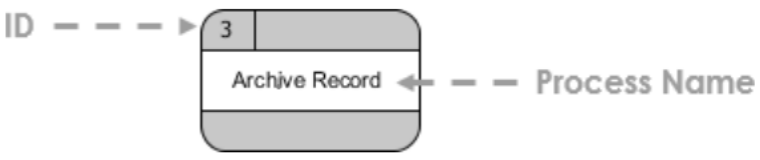


Fig. 3.16 : प्रोसेसेस नोटेशन (Processes Notation)

प्रोसेस उदाहरण (Process Example)

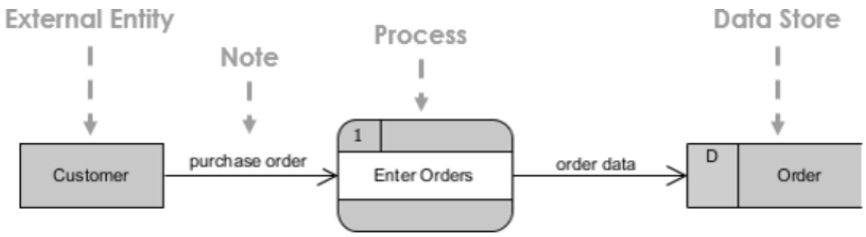


Fig. 3.17 : **प्रोसेस एक्झॅम्पल (Process Example)**

2. **डेटा फ्लो(Data Flow)** म्हणजे प्रोसेस(Process), डेटा(Data) स्टोअर किंवा बाह्य संस्थांमधील डेटा(Data) हालचाल (Data Movement) दर्शवणारे बाण (Arrows). डेटा(Data) प्रवाहाच्या नावांमध्ये साधारणतः संज्ञा वापरली जाते (उदा., "ऑर्डर तपशील").

डेटा(Data) फ्लो उदाहरण:

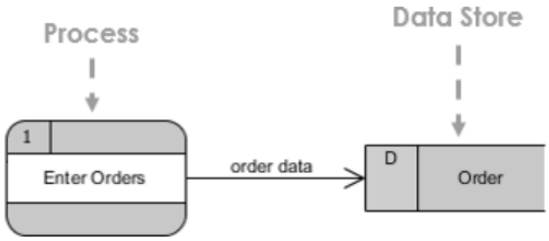


Fig. 3.18 : डेटा फ्लो एक्झॅम्पल (Data Flow Example)

3. **डेटा (Data) स्टोअर** डेटा(Data) स्टोअर म्हणजे डेटाबेस(Data) सारख्या रिपॉझिटरीचे (Repository) चित्रण करणे. गॅन-सरसन नोटेशन मध्ये ओपन-एंडेड आयत (Open-ended Rectangle) वापरले जाते, तर योर्डन / डीमार्को नोटेशन मध्ये समांतर रेषा (Parallel Lines) वापरल्या जातात.

नोटेशन(Notation):

- डेटा(Data) स्टोअरमध्ये डेटा(Data) लिहिता (Write) येतो हे डेटाच्या(Data) 'रायटर' कडून डेटा(Data) स्टोअरमध्ये वाहणाऱ्या फ्लो(Flow) कनेक्टरद्वारे दर्शवले जाते.
- डेटा(Data) स्टोअरमधून डेटा(Data) वाचला (Read) जाऊ शकतो हे डेटा(Data) स्टोअरमधून 'रीडर' कडे वाहणाऱ्या फ्लो कनेक्टरद्वारे दर्शवले जाते.
- डेटा(Data) स्टोअरची उदाहरणे: इन्व्हेंटरी, अकाऊंट्स रिसीव्हेबल, ऑर्डर्स,डेली पेमेंट्स

Fig. 3.19 : डेटा स्टोअर नोटेशन (Data Store Notation)

डेटा(Data) स्टोअर उदाहरण

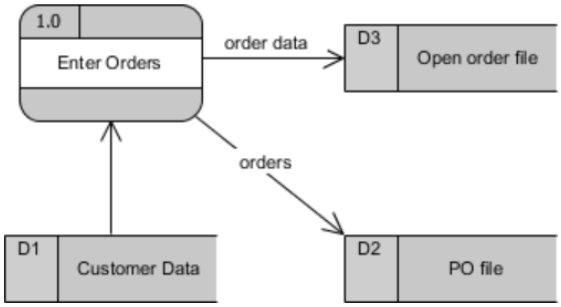


Fig. 3.20 : डेटा स्टोअर एक्झॅम्पल (Data Store Example)

4. **बाह्य संस्था** म्हणजे सिस्टमच्या(System) बाहेरील डेटाचे(Data) स्त्रोत (Source of Data) किंवा गंतव्यस्थाने (Destination of Data). उदा. "ग्राहक", "बँक" इत्यादी. बाह्य संस्था सामान्यतः चौरस (Square) म्हणून दर्शविली जाते. ही सिस्टमच्या(System) बाहेरून येणारा डेटा(Data) किंवा सिस्टममधून(System) बाहेर जाणारा डेटा(Data) कोणाकडून/कोणाकडे जातो हे स्पष्ट करते.

नोटेशन(Notation):

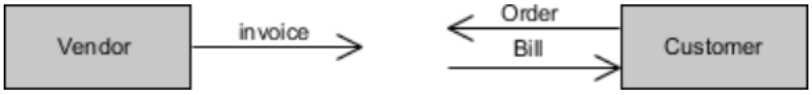


Fig. 3.21 : एक्सटर्नल एंटिटीज नोटेशन (External Entities Notation)

उदाहरण

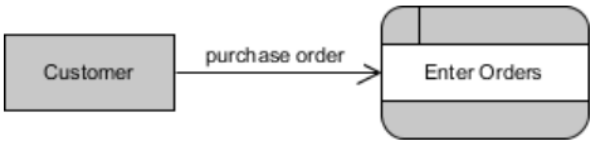


Fig. 3.22 : एक्सटर्नल एंटिटीज एक्झॅम्पल (External Entities Example)

3.4.4. हायरार्किकल लेव्हल्स (Hierarchical Levels)

3.4.4.1. डीएफडी लेव्हल 0 (डीएफडी -0): "डीएफडी संदर्भ" म्हणून देखील ओळखले जाते, या स्तरात, आम्ही आमची सिस्टम)System) मध्यभागी एक प्रोसेस(Inbound)म्हणून ठेवतो आणि बाह्य घटकांपासून इनबाउंड (Process) आणि आउटबाउंड(Outbound)वर लक्ष केंद्रित करतो.

उदाहरण 1- कॉफी शॉपची डी एफ डी लेव्हल 0 (DFD Level 0) :

मुख्य प्रोसेस पी1 (कॉफी शॉप) आहे, लाल बिंदूचक्र आमच्या विश्वासार्ह क्षेत्राचे प्रतिनिधित्व करते (मी त्याबद्दल नंतर बोलेन)

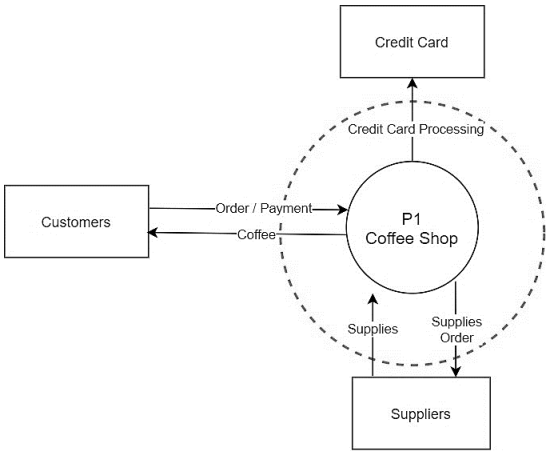


Fig. 3.23 : कॉफी शॉपसाठी डीएफडी लेव्हल 0 (डी एफ डी - 0 (DFD-0)) (DFD level 0 (DFD-0) for Coffee Shop)

उदाहरण 2- डी एफ डी रेलवे आरक्षण सिस्टम

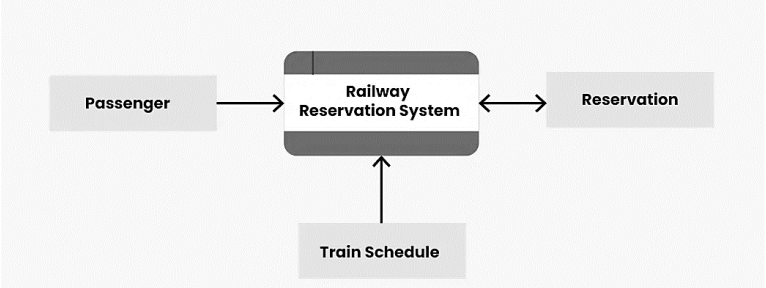


Fig. 3.24: रेल्वे आरक्षण सिस्टमसाठी डीएफडी लेव्हल 0 (डीएफडी -0) (DFD level 0 (DFD-0) for Railway Reservation System)

3.4.4.2. डीएफडी लेव्हल 1 (डीएफडी -1): डीएफडी -0 चे मुख्य व्यवसाय प्रक्रियेत विघटन. हे सहसा सिस्टम च्या मुख्य कार्यात्मक(System)क्षेत्रांचे प्रतिनिधित्व करते

आमच्या उदाहरणात, आम्ही आमचे कॉफी शॉप (पी 1) घेतले आणि पी 1.1 (ऑर्डर मॅनेजमेंट) आणि पी 1.2 (पुरवठा व्यवस्थापन) या 2 मुख्य प्रक्रियांमध्ये वेगळे केले.

जसे आपण पाहू शकता, आमचा विश्वासार्ह झोन अद्याप समान आहे (मुख्य प्रक्रियेचे सर्व "ब्रेक-डाउन" आहेत)

उदाहरण 1- कॉफी शॉपचे डीएफडी लेव्हल 1

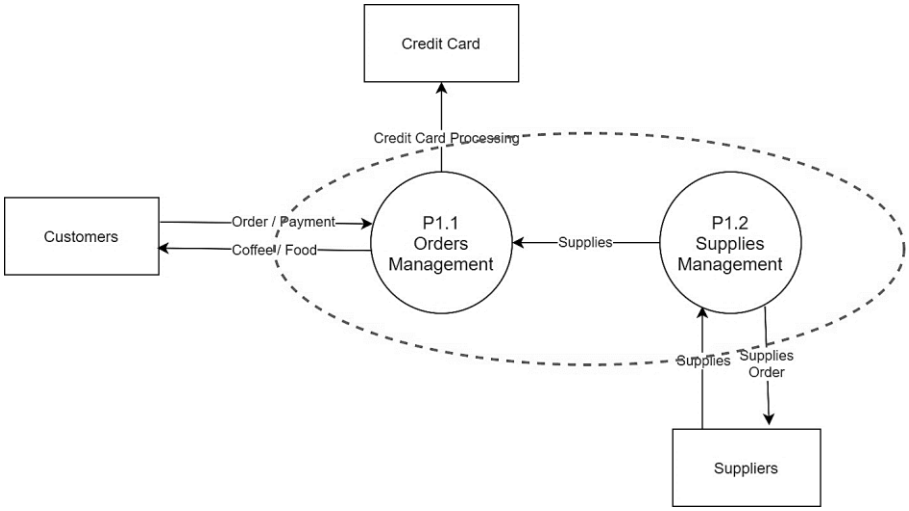


Fig. 3.25: कॉफी शॉपसाठी डीएफडी लेव्हल 1 (डीएफडी -1) (DFD level 1 (DFD-1) for coffee shop)

उदाहरण 2- रेल्वे आरक्षण सिस्टमसाठी डीएफडी लेव्हल 1

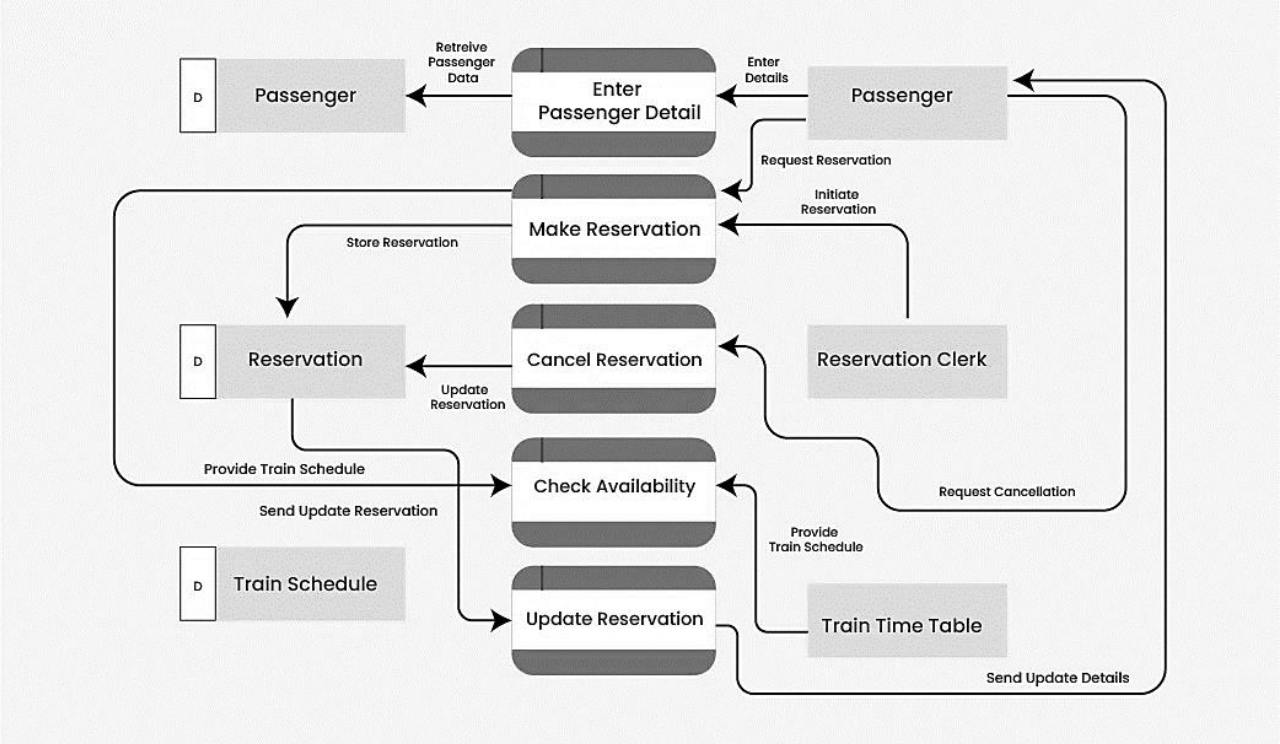


Fig. 3.26: रेल्वे आरक्षण सिस्टमसाठी डीएफडी लेव्हल 1 (डीएफडी -1) (DFD level 1 (DFD-1) for Railway Reservation System)

3.4.4.3. डीएफडी लेव्हल 2 (डीएफडी -2): डीएफडी -1 (डीएफडी -1 ते डीएफडी -0 प्रमाणेच) पासून विघटित होते, तथापि, सामान्यत: या स्तरावर, आपण सेवांची पातळी पाहण्यास सुरवात करतो.

आमच्या उदाहरणात, आम्ही प्रत्येक प्रक्रियेस सेवांमध्ये तोडू.

उदाहरण 1- कॉफी शॉपचे डीएफडी लेव्हल 2

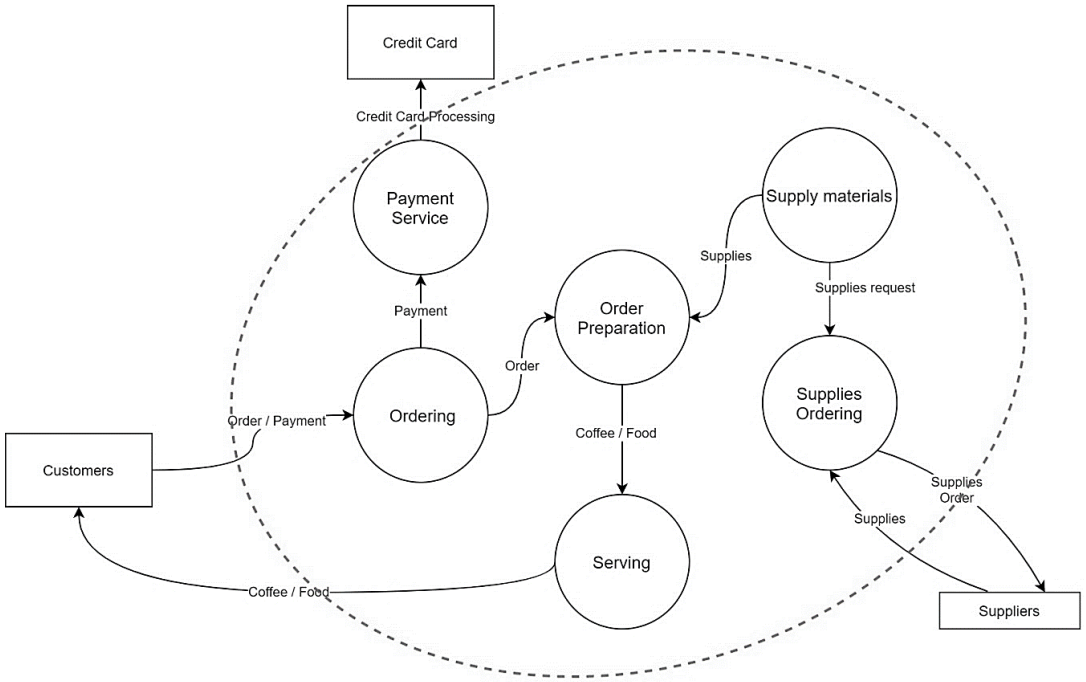


Fig. 3.27: कॉफी शॉपसाठी डीएफडी पातळी 2 (डीएफडी -2)(DFD level 2 (DFD-2) for coffee shop)

उदाहरण 2- रेल्वे आरक्षण सिस्टमसाठी डीएफडी स्तर 2

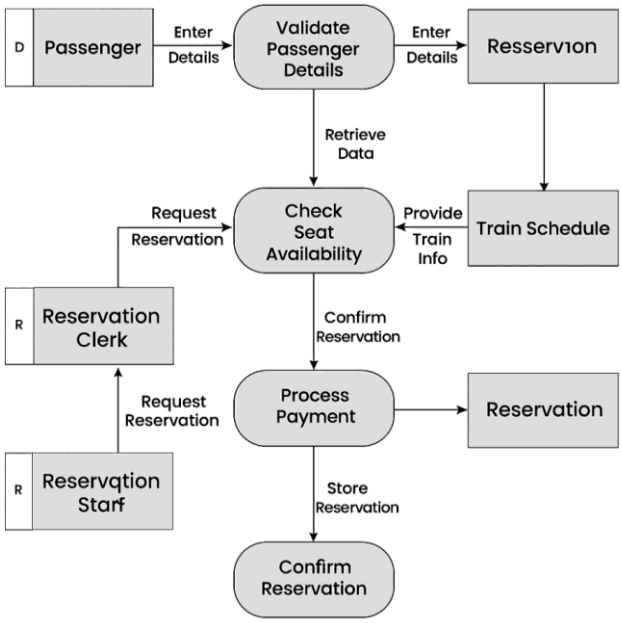


Fig. 3.28 : रेल्वे आरक्षण सिस्टमसाठी डीएफडी स्तर 2 (डीएफडी -2) (DFD level 2 (DFD-2) for Railway Reservation System)

3.4.5. डिज़ाइन नियम (Design Rules)

1. प्रत्येक प्रक्रियेत किमान एक इनपुट फ्लो(input flow) आणि एक आउटपुट फ्लो(ouput flow) असणे आवश्यक आहे.
2. डेटा फ्लो(Data Flow) थेट बाह्य संस्थांना एकमेकांशी जोडू शकत नाही किंवा प्रक्रियांना बायपास करू शकत नाही.
3. प्रत्येक प्रोसेस(Process) ही डेटा मध्ये(Data)काहीतरी बदल (Transformation) करायलाच हवी.
4. सर्व घटकांना (प्रोसेस(Process), डेटा फ्लो(Data Flow), डेटा स्टोअर(Data Store), बाह्य संस्थाExternal) (Organization अनोखी नावे (Unique Names) दिली पाहिजेत.

3.4.6(Structured Flowchart)स्ट्रक्चर्ड फ्लोचार्ट्स .

स्ट्रक्चर्ड फ्लोचार्ट(Structured Flowchart) हा असा फ्लोचार्ट(Flowchart) आहे ज्यामध्ये सर्व प्रोसेस (All Process) आणि निर्णय (Decisions) हे काही मूलभूत स्ट्रक्चर्ड घटकांमध्ये (Basic Structured Elements) बसलेले असले पाहिजेत .स्ट्रक्चर्ड फ्लोचार्ट्स(Structured Flowcharts) प्रामुख्याने अशा परिस्थितीत वापरले जातात जिथे कंट्रोल फ्लो (Control Flow) ची स्पष्टता (Clarity) सर्वांत महत्त्वाची असते.

फ्लोचार्टचे मूलभूत घटक खालील आकृतीत दर्शविलेले आहेत.

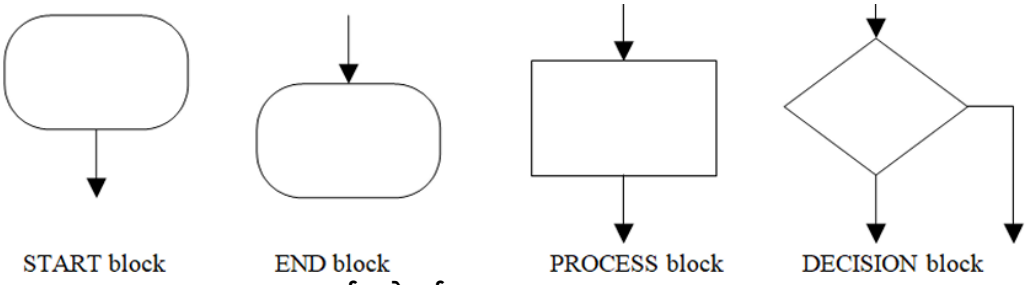


Fig. 3.29 : स्ट्रक्चर्ड फ्लोचार्ट्स घटक (Structured Flowcharts Elements)

- **स्टार्ट ब्लॉक Start Block)):** स्टार्ट ब्लॉक ही एका प्रक्रियेची सुरुवात दर्शवितो. यात नेहमीच एकच आउटपुट असतो. स्टार्ट ब्लॉकला प्रोसीडिंग फ्लोचार्टमध्ये दर्शविलेल्या प्रक्रियेच्या संक्षिप्त वर्णनासह लेबल केले जाते.
- **एंड ब्लॉक :(End Block)**एंड ब्लॉक ही एखाद्या प्रक्रियेचा अंत दर्शवितो. यात नेहमीच एकच इनपुट असतो आणि फ्लोचार्टच्या एकूण प्रक्रियेत, त्याच्या कार्यावर अवलंबून ते एंड किंवा रिटर्न असते.
- **प्रोसेस ब्लॉक**(Process Block): प्रोसेस ब्लॉक(Process Block) हा डेटा(Data)च्या घटकावर केलेल्या काही ऑपरेशनचे (Operation) प्रतिनिधित्व करतो .यामध्ये डेटा वर केलेल्या(Data)प्रक्रियेचे संक्षिप्त वर्णनात्मक लेबल दिलेले असते.
- **डिसिजन ब्लॉक(D** :(ecision Blockडिसिजन ब्लॉक हा नेहमी बायनरी निवड (Binary Choice) करतो. डिसिजन ब्लॉकमधील लेबल(label) हा असा प्रश्न असतो ज्याची स्पष्टपणे दोनच संभाव्य उत्तरे असतात. डिसिजन ब्लॉकमध्ये एक इनपुट आणि दोन आउटपुट असतात. घेतलेल्या निर्णयावर अवलंबून, तर्कप्रवाहाची दिशा) Flow of Logic) दोन आउटपुटला प्रश्नाच्या दोन उत्तरांसह लेबल करून दर्शवली जाते.

स्ट्रक्चर्ड फ्लोचार्टचे मूलभूत घटक:

1. सिक्वेन्स प्रोसेस

सिक्वेन्स प्रोसेस म्हणजे एकामागोमाग एक प्रोसेस अंमलात आणली जाणे. बहुतेक प्रोग्राम्स हे त्यांच्या सर्वोच्च पातळीवर सिक्वेन्सच्या स्वरूपात दर्शविले जातात. यामध्ये कधी कधी लूप असतो, जिथे अंतिम टप्प्यावरून परत सुरुवातीकडे फ्लो परत जातो.

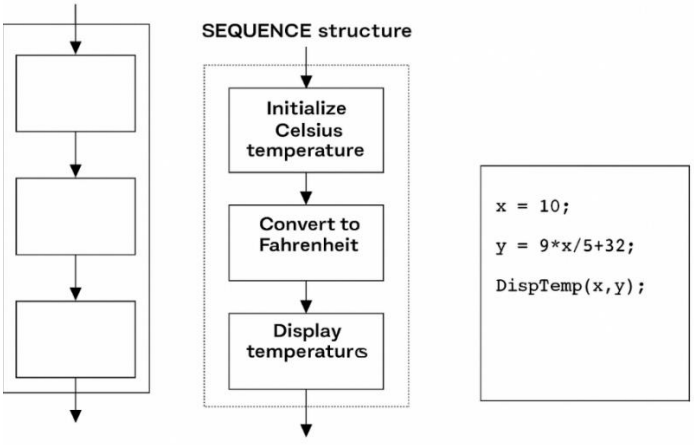


Fig. 3.30 : सिक्वेन्स प्रोसेस (The SEQUENCE process)

2. IF-THEN-ELSE प्रोसेस (Process)

IF-THEN-ELSE प्रोसेस ही बायनरी डिसिजन ब्लॉकला लॉजिकल)Logical)रित्या पूर्ण करते, कारण ती दोन स्वतंत्र प्रोसेससाठी मार्ग पुरवते. बायनरी डिसिजन(Binary Decision) मधून प्रत्येक मार्गावर या दोनपैकी एक प्रोसेस(Process) अंमलात आणली जाते.म्हणजेच, जर अटी)IF) पूर्ण झाली, तर THEN प्रोसेस चालते; अन्यथा ELSE प्रोसेस चालते.

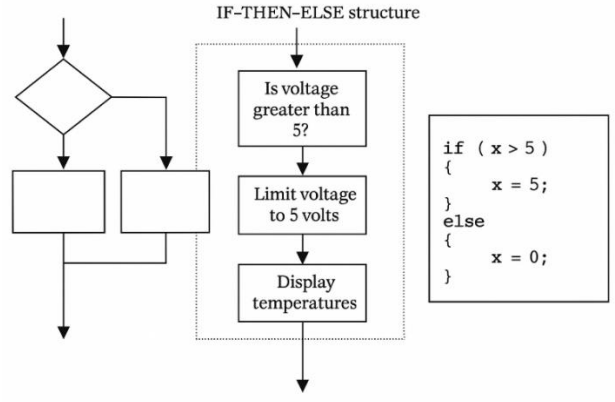


Fig. 3.31 : इफ-देन्-एल्स प्रोसेस (IF-THEN-ELSE Process)

3. WHILE प्रोसेस

WHILE प्रोसेस ही कार्यक्रमामध्ये कंडिशनल लूप स्ट्रक्चरचे(Conditional Loop Structure) प्रतिनिधित्व करण्यासाठी वापरली जाते. लूपमध्ये प्रोसेस(Process) अंमलात आणायची की नाही, हा निर्णय प्रक्रियेची पहिली अंमलबजावणी होण्यापूर्वीच घेतला जातो.जर अट Condition) पूर्ण असेल, तर लूप मधील प्रोसेस पुन्हा पुन्हा चालवली जाते; अन्यथा लूपमधून बाहेर पडले जाते.

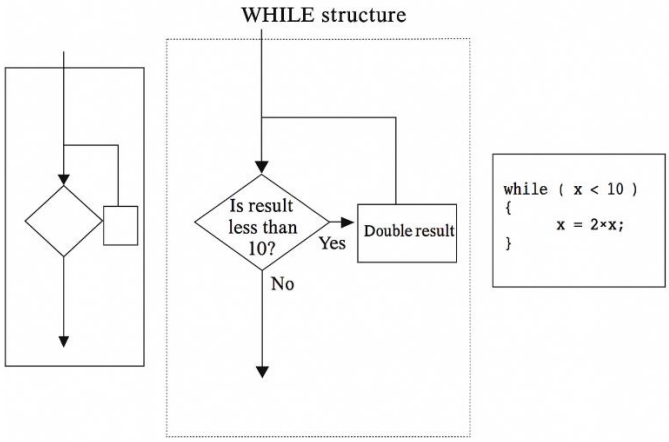


Fig. 3.32: व्हाईल प्रोसेस (WHILE Process)

डिराइव्ह्ड स्ट्रक्चर्स जरी सर्व फ्लोचार्ट्स वरील मूलभूत स्ट्रक्चर्स (सिक्वेन्स(Sequence), इफ(If then else)एल्स-देन-, व्हाईल द्वारे प्रतिनिधित्व करता येतात,

तरी कधी कधी अतिरिक्त स्ट्रक्चर्स — म्हणजेच डिराइव्ह्ड स्ट्रक्चर्स — वापरणे उपयुक्त ठरते.

हे प्रत्येक डिराइव्ह्ड स्ट्रक्चर्स स्वतः वरील मूलभूत स्ट्रक्चर्सपासून बांधले जाऊ शकतात.

1. डूव्हाईल स्ट्रक्चर- (Do-While Structure)

डूव्हाईल स्ट्रक्चर-)Do-While(Structure हे व्हाईल स्ट्रक्चर(While Structure) पेक्षा थोडं वेगळं असतं. डूव्हाईल - मध्ये(Do While Structure)स्ट्रक्चर, लूपमध्ये असलेली प्रोसेस(Process) किमान एकदा तरी अंमलात आणलीच जाते.हे म्हणजे प्रोसेस(Process) एकदा चालवून, त्यानंतरच व्हाईल लूप(While Loop)मध्ये प्रवेश केला जातोव्हाईल . मध्ये(Structure)स्ट्रक्चर मात्र अशी शक्यता असते की प्रोसेस(Process) एकदाही अंमलात न येता लूप थांबू शकतो (जर अट सुरूवातीपासूनच खोटी असेल तरजरी सामान्यतः.(व्हाईल स्ट्रक्चर(While Structure) प्राधान्याने वापरले जाते, तरी काही वेळेस डू(Do While Structure)व्हाईल स्ट्रक्चर- अधिक समजण्यास सोपे असते.

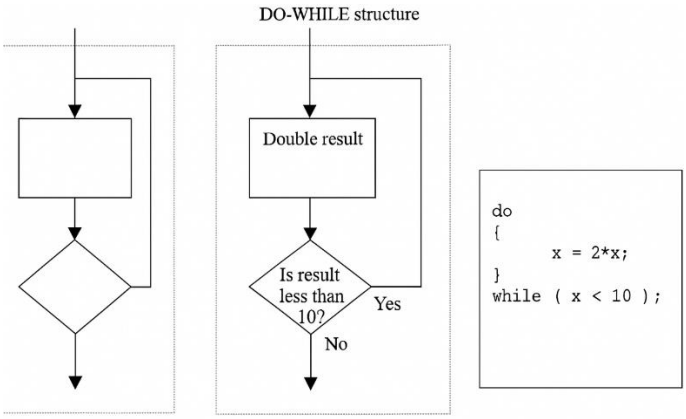


Fig. 3.33: डूव्हाईल स्ट्रक्चर- (Do-While Structure)

2. केस स्ट्रक्चर (Case Structure)

केस स्ट्रक्चर(Case Structure) हे अशा परिस्थितीसाठी उपयुक्त असते जिथे IF-THEN-ELSE स्टेटमेंट्स(Statements) वापरून दोनपेक्षा अधिक पर्यायांची निवड करायची असते .यामध्ये प्रत्येक डिसीजन(DECISION) ब्लॉक एकाच प्रश्नाची किंवा अटीची तुलना करतो, फक्त वेगवेगळ्या पर्यायांशी.उदाहरणार्थ सॉक्सचा रंग कोणता आहे"?" प्रत्येक डिसीजन(DECISION) ब्लॉक मध्ये वेगळा रंग (जसे लाल, हिरवा, निळा इ.तपासला जाईल (.

- जर TRUE असेल, तर फ्लो उजव्या बाजूला जाईल संबंधित प्रोसेस चालवली जाईल.
- जर FALSE असेल, तर फ्लो पुढील डिसीजन(DECISION) ब्लॉक कडे जाईल.

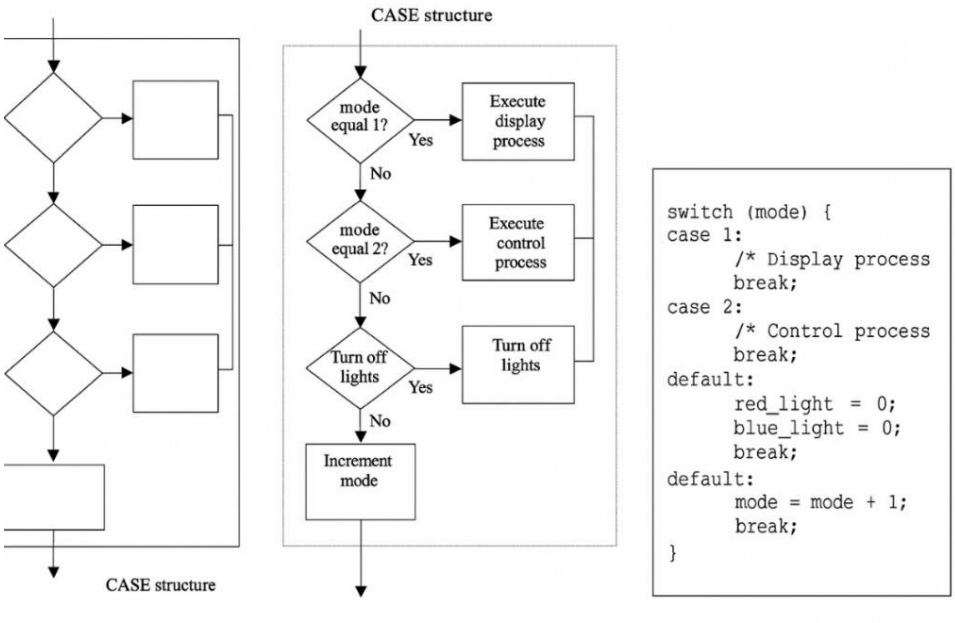


Fig. 3.34: केस स्ट्रक्चर(Case Structure)

3.4.7. डिसिजन टेबल्स (Decision Tables)

डिसिजन टेबल(Decision Tables) टेस्टिंग ही एक सॉफ्टवेअर टेस्टिंग टेक्निक(Testing Technique) आहे, जी सिस्टमचे वर्तन(System) विविध इनपुट(Input) योजनांसाठी चाचणी(Testing) करण्यासाठी वापरली जातेही एक. सिस्टिमॅटिक पद्धत आहे जिथे विविध इनपुट संयोजन आणि त्यांचे संबंधित सिस्टम) वर्तन (System)आउटपुट(तक्त्याच्या स्वरूपात मांडले जाते म्हणूनच.याला कॉजइफेक्ट टेबल-(Cause Effect Table) असेही म्हणतात, जिथे कारणे आणि परिणाम स्पष्टपणे दर्शविले जातात, ज्यामुळे बेहतर टेस्ट कव्हरेज(Test Coverage) मिळतेDecision)डिसिजन टेबल. (Table म्हणजे इनपुट्स(Inputs) विरुद्ध रुल्स(Test Conditions)टेस्ट कंडिशन्स/(Cases)केसेस/(Rules) यांचे

सारणीबद्ध प्रतिनिधित्व. याचा वापर विविध स्थित्यंतरांमध्ये सिस्टमकसे वागेल (System) हे सुस्पष्टपणे समजून घेण्यासाठी केला जातो.

उदाहरण: लॉगिन स्क्रीन(Login Screen)साठी डिसिजन **बेस(Decision Base) टेबल कसे बनवावे**

Fig. 3.35. डिसिजन टेबल लॉगिन स्क्रीनसाठी (Decision Table for Login Screen)

कंडीशन खूप सोप्या प्रकारची आहे जर युजर बरोबर युजरनेम आणि पासवर्ड पुरवतो, तर त्याला होमपेजवर रीडायरेक्ट केले जाईल. जर कोणताही इनपुट चुकीचा असेल (म्हणजे युजरनेम किंवा पासवर्ड), तर एक एरर मेसेज दाखवला जाईल.

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

टी(T)- योग्य वापरकर्ता नाव / पासवर्ड
एफ(F)- चुकीचे युजरनेम/पासवर्ड
ई(E)-एरर संदेश प्रदर्शित केला जातो
एच(H)-होम स्क्रीन प्रदर्शित केली जाते

व्याख्या(Defination):

केस १ - युजरनेम आणि पासवर्ड दोन्ही चुकीचे होते. वापरकर्त्याला एक त्रुटी संदेश दर्शविला जातो.
केस २ - युजरनेम बरोबर होते, पण पासवर्ड चुकीचा होता. वापरकर्त्याला एक त्रुटी संदेश दर्शविला जातो.
केस ३ - युजरनेम चुकीचे होते, पण पासवर्ड बरोबर होता. वापरकर्त्याला एक त्रुटी संदेश दर्शविला जातो.
केस 4 - युजरनेम आणि पासवर्ड दोन्ही बरोबर होते आणि वापरकर्त्याने होमपेजवर नेव्हिगेट केले हे टेस्ट केसमध्ये रूपांतरित करताना, आम्ही 2 सिनारियो)Scenario) तयार करू शकतो
योग्य युजरनेम आणि योग्य पासवर्ड प्रविष्ट करा आणि लॉगिनवर क्लिक करा आणि अपेक्षित परिणाम असा होईल की वापरकर्त्याला होमपेजवर नेव्हिगेट केले पाहिजे

आणि खालील परिस्थितीतील एक

- चुकीचे युजरनेम आणि चुकीचा पासवर्ड टाकून लॉगिनवर क्लिक करा आणि अपेक्षित परिणाम असा होईल की वापरकर्त्याला त्रुटी संदेश मिळाला पाहिजे
- योग्य युजरनेम आणि चुकीचा पासवर्ड प्रविष्ट करा आणि लॉगिनवर क्लिक करा, आणि अपेक्षित परिणाम असा होईल की वापरकर्त्याला एक त्रुटी संदेश मिळाला पाहिजे
- चुकीचे युजरनेम टाकून पासवर्ड दुरुस्त करा आणि लॉगिनवर क्लिक करा, आणि अपेक्षित परिणाम असा होईल की वापरकर्त्याला एक त्रुटी संदेश मिळेल.

3.5. यू एम एल मॉडेलिंग (UML Modelling)

युनिफाइड मॉडेलिंग लँग्वेज(UML Modelling Language) ही सॉफ्टवेअर सिस्टिम्स चे स्पेसिफिकेशन (Specification) , कन्स्ट्रक्शन , आणि डॉक्युमेंटेशन करण्यासाठी स्टँडर्डाइज्ड व्हिज्युअल नोटेशन्स पुरवते. या विभागामध्ये तीन महत्त्वाच्या UML डायग्राम्स यूज-केस डायग्राम(Use Case Diagram), क्लास डायग्राम(Class Diagram), आणि सीक्वेन्स डायग्राम (Swquence Diagram)— यांचा अभ्यास केला जातो. या अभ्यासामध्ये विशेष भर दिला जातो: घटक(Components), नोटेशन(Notation) आणि सॉफ्टवेअर अभियांत्रणामध्ये त्यांचे प्रॅक्टिकल अनुप्रयोग.

3.5.1. यूज-केस डायग्राम्स (Use Case Diagram)

यूज-केस डायग्राम हे युजरचा सिस्टम(System)सोबतचा परस्परसंवाद (User Interaction with System) दर्शवतात आणि फंक्शनल रीक्वायरमेंट्स युजरच्या दृष्टिकोनातून कॅप्चर करतात.

या डायग्राम्स मध्ये ॲक्टर्स(Actors) म्हणजे युजर्स(Users) किंवा एक्सटर्नल सिस्टिम्स, यूज-केसेस(Use Cases) म्हणजे सिस्टमच्या(System) फंक्शनॅलिटीज, आणि त्यांच्यामधील रिलेशनशिप्स ओळखल्या जातात.

यूज-केस डायग्राम्स हे रिप्वायरमेंट्स ॲनालिसिसच्या टप्प्यावर फाउंडेशनल असतात जेणेकरून सिस्टमचा(System) स्कोप आणि स्टेकहोल्डरच्या गरजा परिभाषित करता येतात.केस डायग्राम नोटेशन वापरा.

यूज-केस डायग्राम नोटेशन्स (Use Case Diagram Notations)

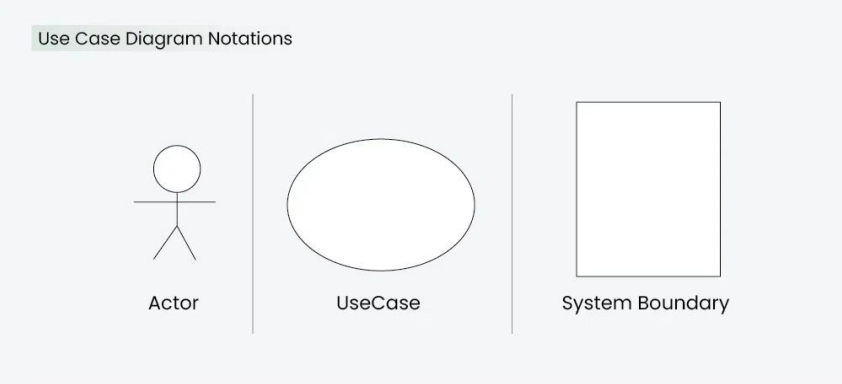


Fig. 3.36: यूज-केस डायग्राम नोटेशन्स (Use Case Diagram Notations)

1. **ॲक्टर (Actor)**
   ॲक्टर (Actor) म्हणजे सिस्टमसोबत(System) परस्परसंवाद करणाऱ्या बाह्य घटक (External Entities).यामध्ये युजर्स(Users), इतर सिस्टिम्स(Systems), किंवा हार्डवेअर डिव्हायसेस समाविष्ट असू शकतात. यूज-केस(Use case) डायग्रामच्या संदर्भात, ॲक्टर हे यूज-केसेस(Use Cases) सुरू (Initiate) करतात आणि त्याचे आउटकम्स (Outcomes) प्राप्त करतात.
2. **यूज-केस (Use Case)**
   यूज-केसेस(Use Cases) म्हणजे प्लेमधील दृश्यांसारखे (Scenes in a Play). ते सिस्टम(System) काय करू शकते हे विशिष्ट फंक्शनॅलिटीज म्हणून प्रतिनिधित्व करतात.
3. **सिस्टिम बाउंडरी (System Boundary)**
   सिस्टिम बाउंडरी ही सिस्टमच्या(System) स्कोप किंवा मर्यादा (Scope or Limits) चे व्हिज्युअल प्रतिनिधित्व आहे. हे सिस्टममध्ये(System) काय अंतर्भूत आहे (Inside the System) आणि सिस्टमबाहेर(System) काय आहे (Outside) हे स्पष्टपणे दर्शवते.

3.5.1.2 उदाहरण: ग्रंथालय व्यवस्थापन सिस्टम Libraray Managment)System)

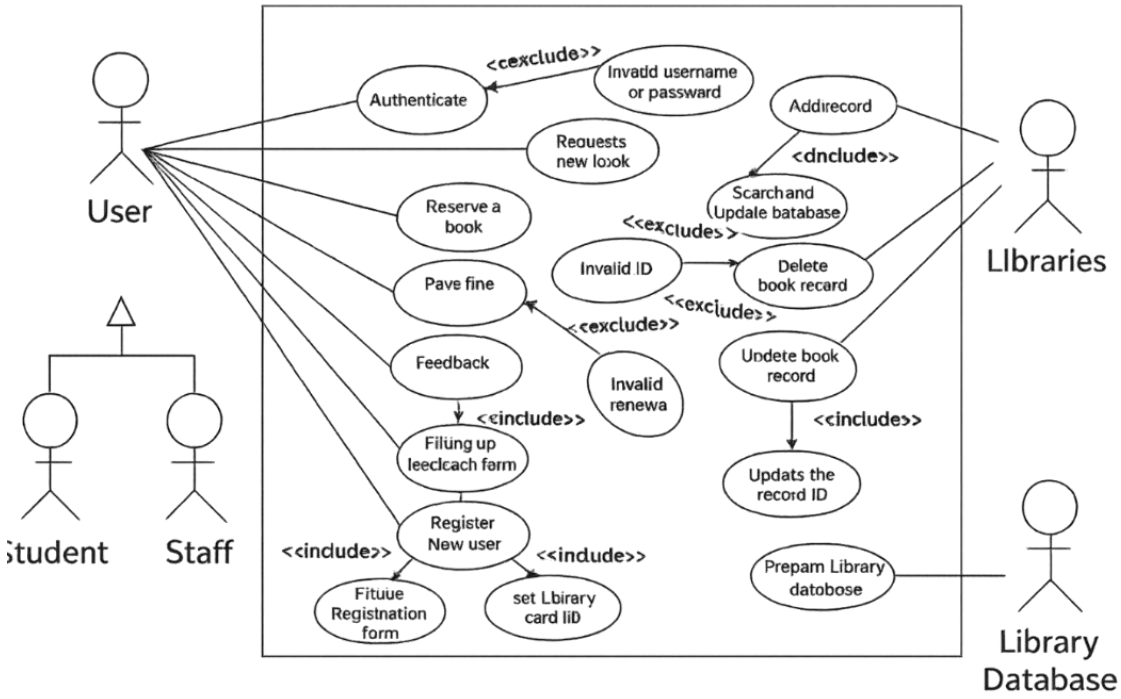


Fig. 3.37: लायब्ररी मॅनेजमेंट सिस्टीमसाठी केस डायग्राम)Case Diagram for Library Managment System(

यूज-केस डायग्रामसाठी टेक्स्टुअल वर्णन

1. **ॲक्टर्स (Actors):**
   1. युजर — स्टाफ किंवा स्टुडंट (User- Staff or Student)
   2. लायब्रेरियन(Librarian)
2. **यूज-केसेस (Use cases):**
   1. रजिस्टर न्यू युजर (Register New User)
   2. इश्यू लायब्ररी कार्ड (Issue Library Card)
   3. रिक्वेस्ट न्यू बुक (Request New Book)
   4. रिझर्व बुक (Reserve Book)
   5. रिन्यू बुक (Renew Book)
   6. पे फाईन (Pay Fine)
   7. फिल फीडबॅक फॉर्म (Fill Feedback Form)
   8. मॅनेज रेकॉर्ड्स (Manage Records)
   9. डिलीट रेकॉर्ड्स (Delete Records)
   10. अपडेट डेटाबेस (Update Database)
3. **सिस्टिम बाउंडरी (System Boundary):**
   सिस्टिम बाउंडरी मध्ये वरील सर्व यूज-केसेस समाविष्ट असतील.

लायब्ररी मॅनेजमेंट सिस्टिमसाठी यूज-केस डायग्राम — संक्षिप्त माहिती (Information)

येथे आपण लायब्ररी मॅनेजमेंट सिस्टिमसाठी यूज-केस डायग्राम डिझाईन कसा करायचा हे समजून घेणार आहोत. खालील काही प्रमुख सीनारिओज या सिस्टिममध्ये आहेत:

1. **नवीन युजर रजिस्ट्रेशन (New User Registration):**
   - युजर जेव्हा स्वतःला नवीन युजर म्हणून रजिस्टर करतो, तेव्हा तो स्टाफ किंवा स्टुडंट म्हणून लायब्ररी सिस्टिममध्ये नोंदवला जातो. रजिस्ट्रेशन फॉर्म भरून युजरची नोंदणी होते.

- नोंदणीनंतर लायब्रेरियनकडून युजरला लायब्ररी कार्ड इश्यू केले जाते, ज्यावर युजरचा ID दिलेला असतो.

2. **नवीन बुक रिक्वेस्ट (New Book Request)**
   - लायब्ररी कार्ड मिळाल्यानंतर, युजर आपल्या गरजेनुसार नवीन बुकची रिक्वेस्ट करू शकतो.
3. **बुक रिझर्वेशन (Book Reservation)**
   - रिक्वेस्ट केल्यानंतर, बुक रिझर्व्ह केली जाते म्हणजेच इतर युजर ती बुक रिक्वेस्ट करू शकत नाहीत.
4. **बुक रिन्यू (Book Renew)**
   - युजर बुक रिन्यू करू शकतो — म्हणजेच नवीन ड्युएट मिळवू शकतो, जर त्याने बुक रिन्यू केली असेल.
5. **फाईन पेमेंट (Fine Payment)**
   - जर युजरने बुक ड्यु पूर्वी परत केली नाही, किंवा रिन्यू करायचे विसरला, तर बुक ओव्हरड्यू होते आणि युजरला फाईन भरावी लागते.
6. **फीडबॅक फॉर्म भरणे (Fill Feedback Form)**
   - युजर फीडबॅक फॉर्म भरू शकतो, जर त्याला फीडबॅक द्यायचा असेल.
7. **लायब्रेरियनचे मुख्य कार्य (Main Work of Librarian)**
   a. **रेकॉर्ड्स मॅनेज करणे (Maanage Record) -** लायब्रेरियन प्रत्येक वेळी बुक इश्यू, बुक रिटर्न, किंवा फाईन पेमेंट याबाबत डेटाबेस(Data) अपडेट करतो.
   b. **रेकॉर्ड्स डिलीट करणे (Delete Record) -** जर विद्यार्थी कॉलेज सोडतो किंवा पासआउट होतो, तर त्याचे रेकॉर्ड डिलीट केले जाते.तसेच, जर बुक लायब्ररीतून काढून टाकली गेली असेल, तर त्या बुकचा रेकॉर्ड पण डिलीट केला जातो.
   c. **डेटाबेस अपडेट करणे (Update Database) -** हे लायब्रेरियनचे महत्त्वाचे कार्य आहे.

3.5.2. क्लास डायग्राम्स (Class Diagrams)

क्लास डायग्राम्स(Class Diagrams) हे सिस्टम ची स्थिर रचना(System)(Static Structure of System) दर्शवतात. यामध्ये क्लासेस (Classes), ॲट्रिब्यूट्स (Attributes), मेथड्स (Methods), आणि रिलेशनशिप्स (Relationships) दाखवल्या जातात.क्लास डायग्राम्स(Class Diagrams) हे ऑब्जेक्ट-ओरिएंटेड डिझाइन (Object-Oriented Design) चा कणा (Backbone) असतात.त्यामुळे सिस्टमचे घटक(System), त्यांची वैशिष्ट्ये आणि परस्पर संबंध स्पष्टपणे प्रतिनिधित्व केले जातात.

3.5.2.1. कोर घटक (Core Components)

1. **क्लास :** क्लास म्हणजे ऑब्जेक्टसाठी एक ब्लूप्रिंट. क्लासचे प्रदर्शन तीन भागांमध्ये विभागलेले असते:
   - नेम (Name): बोल्ड, सेंटरमध्ये लिहिले जाते. उदा. विद्यार्थी.
   - ॲट्रिब्यूट्स (Attributes): व्हेरिएबल्स दर्शवतात. उदा. studentId: String.
   - ऑपरेशन्स (Operations) :मेथड्स दर्शवतात. उदा. calculateGPA()
2. **रिलेशनशिप्स (Relationships)**
   - ॲसोसिएशन(Association): सॉलिड लाईन द्वारे क्लासेस जोडले जातात. उदा. Student attends Course
   - ॲग्रीगेशन(Aggregation) : हॉलो डायमंड वापरून "has-a" रिलेशनशिप दर्शवली जाते. उदा. Department —◇— Professor
   - कॉम्पोझिशन(Composition): फिल्ड डायमंड वापरून मजबूत मालकी दर्शवली जाते. उदा. Car —◆— Engine
   - इनहेरिटन्सI)n (heritance: हॉलो ॲरो वापरून जनरलायझेशन दर्शवली जाते. उदा. Vehicle ←◁— Car
   - डिपेंडन्सी (Dependency): डॅश्ड ॲरो वापरून तात्पुरता वापर दर्शवला जातो. उदा. Student ⟶ PaymentService
3. **मल्टिप्लिसिटी (Multiplicity) :** मल्टिप्लिसिटी म्हणजे कार्डिनॅलिटी (Cardinality) दर्शवते — म्हणजे एखादा क्लास दुसऱ्या क्लासशी किती वेळा रिलेट होतो. उदा.: 1..* (एक किंवा अधिक), 0..1 (शून्य किंवा एक).

3.5.5.2 उदाहरण: विद्यापीठ नोंदणी सिस्टम (Univerisity Reservation System)

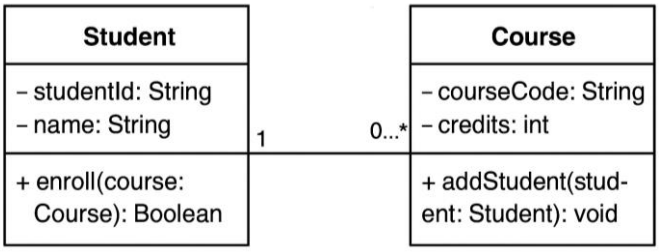


Fig. 3.38: विद्यापीठ नोंदणी सिस्टम (Univerisity Reservation System)

3.5.3. सीक्वेन्स डायग्राम्स(Sequence Diagram)

सीक्वेन्स डायग्राम्स हे एखाद्या विशिष्ट सीनारिओमध्ये ऑब्जेक्ट्समध्ये(Object) घडणाऱ्या कालानुक्रमिक इंटरॲक्शनचे व्हिज्युअलायझेशन करतात. हे रनटाईम दरम्यान डायनॅमिक बिहेवियर चे सविस्तर वर्णन करण्यासाठी वापरले जातात.

3.5.3.1. कोर घटक (Core Components)

1. **लाइफलाईन्स(LifeLines):**
   - लाइफलाईन्स म्हणजे ऑब्जेक्ट्स किंवा ॲक्टर्स दर्शवणाऱ्या उभ्या तुटक रेषा. उदा. Student, Database
   - ऍक्टिवेशन बार्स (Activation Bars): लाइफलाईनवर असलेले आयताकृती बार्स जे मेथड एक्झिक्युशनचा कालावधी दर्शवतात.
2. **मेसेजेस (Messages)**
   - सिंक्रोनस मेसेज(Synchronous Message): भरलेले हेड असलेला सॉलिड ॲरो उदा.: validateLogin()
   - असिंक्रोनस मेसेज(Asynchronous Message) ओपन हेड असलेला सॉलिड ॲरो (उदा.: sendNotification())
   - रिटर्न मेसेज ओपन हेड असलेला तुटक ॲरो वापर: एखाद्या मेथडच्या एक्झिक्युशननंतर रिटर्न व्हॅल्यू दाखवण्यासाठी. उदा.: return: success
3. **कॉम्बाइन्ड फ्रॅगमेंट्स (Combined Fragments)**
   - लूप (Loop)
     - पुनरावृत्ती होणाऱ्या इंटरॲक्शनसाठी वापरले जाते.
     - उदा.: अनेक आयटेम्सवर प्रोसेस(Process) करणे.
   - अल्ट/एल्स
     - कंडिशनल लॉजिक दर्शवण्यासाठी वापरले जाते.
     - उदा.: जर वैध असेल तर पुढे जा, अन्यथा त्रुटी दाखवा.
   - ऑप्ट
     - पर्यायी इंटरॲक्शनसाठी वापरले जाते.
     - उदा.: डिस्काउंट लागू करणे.

3.5.3.2. उदाहरण: एटीएम विथड्रॉव्हल प्रोसेस (ATM Withdrawal Process)

ॲक्टर्स(Actors) / ऑब्जेक्ट्स(Objects)

1. युजर
2. एटी एम मशीन
3. बँक सिस्टिम

सीक्वेन्स फ्लो

1. User → ATM: insertCard() → युजर कार्ड घालतो (insertCard).
2. ATM → Bank: validateCard()→ एटीएम बँकेला कार्ड व्हॅलिडेशनसाठी विनंती पाठवतो.
3. Bank → ATM: return valid → बँक एटीएमला "valid" (वैध) उत्तर पाठवते.

4. ATM → User: enterPin()→ एटीएम युजरला पिन (PIN) टाकण्यास सांगतो.
5. User → ATM: pin→ युजर पिन एटीएममध्ये एंटर करतो.
6. ATM → Bank: verifyPin()→ एटीएम बँकेला पिन सत्यापनासाठी (verify) विनंती पाठवतो.
7. Bank → ATM: return success → बँक "success" (यशस्वी) उत्तर पाठवते.
8. ATM → User: selectAmount()→एटीएम युजरला रक्कम निवडण्यास सांगतो (selectAmount).

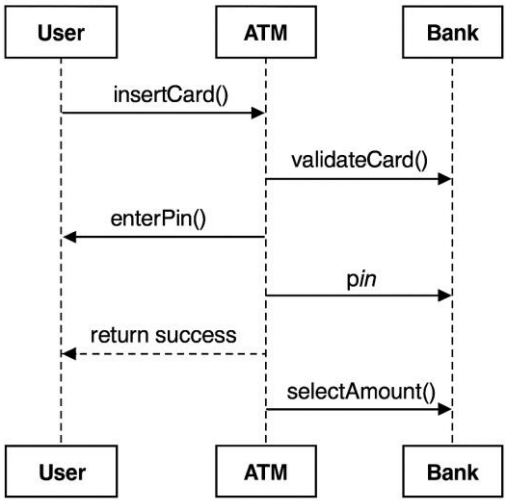


Fig. 3.39: एटी एम मधून पैसे काढण्याची प्रोसेस (ATM Withdrawal Process)

3.5.3.3 डायग्राम्समधील इंटीग्रेशन (Integration within diagram)

1. यूज-केस → क्लास :

- ॲक्टर्स (Actors) → क्लासेस मध्ये रूपांतरित होतात. उदा.: Student (ॲक्टर्स(Actors)) → Student (क्लासेस).
- यूज-केसेस → मेथड्स मध्ये रूपांतरित होतात.उदा.: enroll(), payFees().

2. क्लास → सीक्वेन्स (Class → Sequence)

- क्लास ऑपरेशन्स → लाइफलाईन्समधील मेसेजेस मध्ये मॅप होतात. उदा.: Student.enroll() → Message → Course.enrollStudent().

3.5.3.4. इंटीग्रेशनसाठी आवश्यक फोकस

- यूज-केस: ॲक्टर आणि यूज-केस यांच्यातील योग्य रिलेशनशिप्स. सिस्टिम बाउंडरीज (System Boundaries) स्पष्टपणे ठरवणे.
- क्लास: ॲग्रीगेशन आणि कॉम्पोझिशन यांचा योग्य उपयोग करणे.
- सीक्वेन्स: अचूक मेसेज ऑर्डरिंग. कॉम्बाइन्ड फ्रॅगमेंट्सचा योग्य वापर.

3.6. टेस्टिंग (Testing)

3.6.1 सॉफ्टवेअर टेस्टिंगचा अर्थ आणि उद्देश)Meaning and Purpose of Software Testing)

सॉफ्टवेअर टेस्टिंग ही एक सिस्टिमॅटिक प्रोसेस आहे, जिच्या माध्यमातून सॉफ्टवेअर ॲप्लिकेशन्सचे मूल्यांकन केले जाते. यामध्ये खालील बाबींचा समावेश होतो: फंक्शनॅलिटीची पडताळणी, त्रुटी किंवा दोष ओळखणे, आणि स्पेसिफाइड रीक्वायरमेंट्सशी सुसंगती सुनिश्चित करणे. सॉफ्टवेअर टेस्टिंगचा प्राथमिक उद्देश:सॉफ्टवेअर फेल्युअर्सशी संबंधित धोके कमी करणे. अंतिम उत्पादन हे वापरकर्त्याच्या अपेक्षा आणि ऑपरेशनल स्टँडर्ड्स पूर्ण करते याची पडताळणी करणे.

3.6.1.1 मुख्य उद्दिष्टे (Key Objectives)

- डिफेक्ट आयडेंटिफिकेशन: संरचित टेस्ट केसेसच्या माध्यमातून कोडिंग एरर्स, डिझाईन फ्लॉज, आणि रिक्वायरमेंट गॅप्स उघड करणे.
- व्हॅलिडेशन आणि व्हेरिफिकेशन:

व्हॅलिडेशन : सॉफ्टवेअर जसे अपेक्षित आहे तसे कार्य करते हे सुनिश्चित करणे.
व्हेरिफिकेशन : सॉफ्टवेअरचे डिझाईन स्पेसिफिकेशन्सशी पालन सुनिश्चित करणे.

- रिस्क मिटिगेशन:
  विकासाच्या सुरुवातीच्या टप्प्यात समस्या शोधून, पोस्ट-डिप्लॉयमेंटच्या महागड्या फेल्युअर्स टाळणे.
- क्वालिटी ॲशुरन्स:
  परफॉर्मन्स बेंचमार्किंग आणि कॉम्प्लायन्स चेक्स च्या माध्यमातून विश्वसनीयता , सुरक्षा , आणि वापरयोग्यता सुनिश्चित करणे.

3.6.2 टेस्टिंग मेथडॉलॉजीज (Testing Methodologies)

3.6.2.1 ब्लॅक-बॉक्स टेस्टिंग (Black-Box Testing)

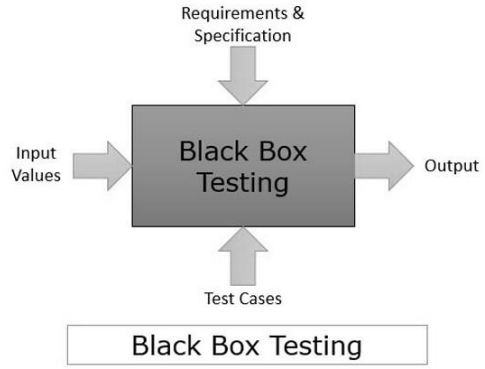


Fig. 3.40: ब्लॅक-बॉक्स चाचणी (Black-Box Testing)

1. आंतरर्गत सिस्टम(System) डिझाईन या प्रकारात गृहित धरले जात नाही.
2. टेस्ट्स या रिक्वायरमेंट्स आणि फंक्शनॅलिटी आधारित असतात.
3. टेस्टर ला सिस्टम(System) आर्किटेक्चरची माहिती(Information) नसते, आणि त्याला सोर्स कोड सुद्धा प्रवेशासाठी उपलब्ध नसतो.
4. टेस्टर्स हे सिस्टमच्या(System) युजर इंटरफेससोबत(Interface) इंटरॲक्ट करतात, आणि इनपुट्स प्रदान करतात.
5. आउटपुटचे परीक्षण केले जाते, पण इनपुटवर कसा आणि कुठे प्रोसेस केला जातो हे हे माहिती (Information) .नसते

ॲडव्हान्टेजेस (Advantages)

1. मोठ्या सिस्टिम्सवर वापरल्यास प्रभावी
2. टेस्टर आणि डेव्हलपर हे एकमेकांपासून स्वतंत्र असल्यामुळे, टेस्टिंग संतुलित आणि पूर्वग्रहमुक्त होते.
3. टेस्टर हा नॉनटेक्निकल- असू शकतो.
4. टेस्टरला सिस्टमचे सविस्तर फंक्शनल नॉलेज(System) असण्याची गरज नाही.
5. टेस्टिंग एंड युजरच्या-दृष्टिकोनातून केले जाते, कारण एंड युजर नेच शेवटी सिस्टमस्वीकारायची (System) (कधी कधी ही टेस्टिंग टेक्निक "ॲक्सेप्टन्स टेस्टिंग.म्हणून ओळखली जाते ")
6. फंक्शनल स्पेसिफिकेशन्समधील अस्पष्टता आणि विरोधाभास ओळखण्यास मदत होते.
7. टेस्ट केसेस फंक्शनल स्पेसिफिकेशन्स पूर्ण होताच डिज़ाइन(Design) करता येतात.

डिसॲडव्हान्टेजेस (Disadvantages)

1. स्पष्ट फंक्शनल स्पेसिफिकेशन्स नसल्यास टेस्ट केसेस डिज़ाइनकरणे आव्हानात्मक (Design) असते.
2. टेस्ट केसेस जर स्पेसिफिकेशन्सवर आधारित डेव्हलप केल्या नसतील, तर कठीण इनपुट्स ओळखणे कठीण असते.
3. मर्यादित टेस्टिंग वेळेत सर्व संभाव्य इनपुट्स ओळखणे कठीण असते. यामुळे टेस्ट केसेस लिहिणे हळू आणि कठीण होऊ शकते.

4. टेस्टिंग प्रोसेसमध्ये अजाणत्या मार्गांचे राहण्याची शक्यता असते.
5. प्रोग्रॅमरने आधीच केलेल्या टेस्ट्स परत पुन्हा करण्याची अधिक शक्यता असते.

3.6.2.2 व्हाइट बॉक्स टेस्टिंग (White-Box Testing)

व्हाइट बॉक्स टेस्टिंग ही ऍप्लिकेशनच्या कोडच्या अंतर्गत लॉजिकच्या ज्ञानावर आधारित असते. ही ग्लास बॉक्स, क्लिअर बॉक्स किंवा ओपन बॉक्स टेस्टिंग म्हणूनही ओळखली जाते. या प्रकारासाठी सॉफ्टवेअरचे आंतरर्गत कार्य माहित असणे आवश्यक आहे. कोडच्या अंतर्गत कार्यपद्धतीचे ज्ञान असावे लागते.

मुख्य वैशिष्ट्ये:

1. कोडच्या अंतर्गत लॉजिक आणि स्ट्रक्चरचे सविस्तर तपासणी(Structure)
2. सोर्स कोडमध्ये पाहून कोणता कोड युनिट चुकीचे वागतो ते शोधणे.
3. ब्लॅकबॉक्स टेस्टिंगमध्ये वगळलेले पैलू- व्हाइटबॉक्स टेस्टिंगमध्ये चाचणी केली जाते-.

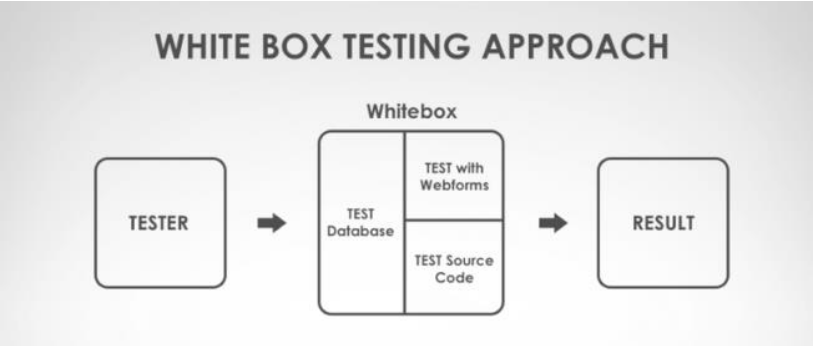


Fig. 3.41 : व्हाइट बॉक्स टेस्टिंग (White-Box Testing)

ॲडव्हान्टेजेस - व्हाइट) बॉक्स टेस्टिंग-White-Box Testing)

1. कोणत्या प्रकारचा डेटा ऍप्लिकेशन टेस्टिंगमध्ये प्रभावी (Data)ठरेल हे शोधणे खूप सोपे होते.
2. एक्स्ट्रा कोडच्या ओळी (अवांछित) काढून टाकता येतात, ज्यामुळे लपलेले दोष (Hidden Defects) समोर येतात.
3. कोड ऑप्टिमाइझ (Optimize) करण्यात मदत होते.
4. टेस्टरला कोडबद्दल माहितीअसल्यामुळे (Information), टेस्ट केसेस लिहिताना जास्तीत जास्त कव्हरेज (Maximum Coverage) साध्य होते.

डिसॲडव्हान्टेजेस - व्हाइट) बॉक्स टेस्टिंग-White-Box Testing)

1. व्हाइट-बॉक्स टेस्टिंग करण्यासाठी कौशल्य असलेला टेस्टर आवश्यक असल्यामुळे, कॉस्ट वाढते.
2. प्रत्येक पाथ आणि कंडिशन टेस्ट करण्यासाठी फुल रेंज इनपुट्स तयार करावी लागतात, त्यामुळे ही पद्धत टाइम-कंझ्युमिंग आहे.
3. सर्व कंडिशन्स टेस्ट करणे वास्तविक दृष्ट्या शक्य नसते, त्यामुळे काही कंडिशन्स टेस्ट होऊ शकत नाहीत.
4. स्पेशलाइज्ड टूल्स जसे की कोड ॲनालायझर्स आणि डिबगिंग टूल्स आवश्यक असतात.

ब्लॅक बॉक्स टेस्टिंग आणि व्हाईट बॉक्स टेस्टिंग मधील फरक :

ब्लॅकबॉक्स टेस्टिंग	व्हाईटबॉक्स टेस्टिंग
कोडचे आंतरर्गत डिझाईन किंवा इम्प्लिमेंटेशन टेस्टरला माहित नसते.	कोडचे आंतरर्गत डिझाईन किंवा इम्प्लिमेंटेशन टेस्टरला माहित असते.
मुख्यतः सॉफ्टवेअर टेस्टरद्वारे केली जाते.	मुख्यतः सॉफ्टवेअर डेव्हलपरद्वारे केली जाते.
इम्प्लिमेंटेशनचे ज्ञान आवश्यक नाही.	इम्प्लिमेंटेशनचे ज्ञान आवश्यक आहे.
हे "External Software Testing" म्हणून ओळखले जाते.	हे "Internal Software Testing" म्हणून ओळखले जाते.

ब्लॅकबॉक्स टेस्टिंग	व्हाईटबॉक्स टेस्टिंग
SRS डॉक्युमेंटवर आधारित टेस्टिंग सुरू करता येते.	Detailed Design डॉक्युमेंटवर आधारित टेस्टिंग सुरू करता येते.
प्रोग्रॅमिंगचे ज्ञान आवश्यक नाही.	प्रोग्रॅमिंग भाषेचे ज्ञान आवश्यक आहे.
हे Behavioural Testing चा प्रकार आहे.	हे Logical Testing चा प्रकार आहे.
Higher Level Software Testing साठी लागू होते.	Lower Level Software Testing साठी लागू होते.
Closed Testing म्हणून देखील ओळखले जाते.	Clear Box Testing म्हणून देखील ओळखले जाते.
Types: – Functional Testing – Non-functional Testing – Regression Testing	Types: – Path Testing – Loop Testing – Condition Testing

3.6.3. स्टॅटिक आणि डायनॅमिक टेस्टिंग (Static and Dynamic Testing)

3.6.3.1. स्टॅटिक टेस्टिंग (Static Testing)

स्टॅटिक टेस्टिंग म्हणजे सॉफ्टवेअरचे वर्क प्रॉडक्ट्स (Work Products) जसे की कोड, डिझाईन, डॉक्युमेंट्स इ. कार्यक्रम चालू न करता (Without Executing the Program) तपासण्याची प्रोसेस हे. विकास चक्राच्या सुरुवातीला (Early in the Development Cycle) केले जाते जेणेकरून— कोडिंग पूर्ण होण्याआधीच त्रुटी (Errors) शोधता येतात.

उद्देश(Purpose):

1. डॉक्युमेंट्स आणि कोडमधील त्रुटी लवकर शोधणे (Find Defects Early in Documents and Code).
2. कोडिंग स्टँडर्ड्स (Coding Standards) आणि डिझाईन गाईडलाइन्स (Design Guidelines) पालन होत आहे का ते सुनिश्चित करणे.
3. सॉफ्टवेअरच्या लॉजिक आणि स्ट्रक्चरची पडताळणी (Verify Logic and Structure of Software).

ॲडव्हान्टेजेस(Advantages):

1. त्रुटी लवकर शोधल्यामुळे खर्च कमी होतो**.**
2. कोड चालवण्याची गरज नाही**.**
3. कोडची गुणवत्ता आणि एकसंधता सुधारण्यास मदत होते**.**
4. डॉक्युमेंट्स व्हेरिफाय करण्यासाठी उपयुक्त**.**

3.6.3.2. डायनॅमिक टेस्टिंग (Dynamic Testing)

डायनॅमिक टेस्टिंग मध्ये कोड चालवून (Executing the Code) त्याची फंक्शनॅलिटी (Functionality) आणि परफॉर्मन्स (Performance) व्हॅलिडेट केली जाते. हे कोडिंगनंतर (After Coding) केले जाते जेणेकरून— सॉफ्टवेअर अपेक्षेनुसार काम करते की नाही हे तपासता येते.

उद्देश (Purpose):

1. सॉफ्टवेअरचा वर्तन रिअलटाइममध्ये पडताळणे-.
2. फंक्शनल आणि नॉनफंक्शनल रीक्वायरमेंट्स-.
3. रनटाइम एरर्स, मेमरी लिक्स, आणि परफॉर्मन्स इश्यूज तपासणे.

ॲडव्हान्टेजेस(Advantages):

1. सॉफ्टवेअरच्या Actual वर्तनाची पडताळणी.
2. रनटाइम एरर्स पकडतो, जे स्टॅटिक टेस्टिंगमध्ये सापडत नाहीत.
3. सॉफ्टवेअर रिअल एन्व्हायरमेंटमध्ये काम करते का ते तपासते.
4. परफॉर्मन्स, रिस्पॉन्स टाइम इत्यादी मोजण्यास मदत होते.

फीचर (Feature)	स्टॅटिक टेस्टिंग (Static Testing)	डायनॅमिक टेस्टिंग (Dynamic Testing)
कधी केली जाते (When Performed)	कोड चालवण्याच्या आधी (Before Code Execution)	कोड चालवताना किंवा नंतर (During/After Code Execution)
उद्देश (Purpose)	डॉक्युमेंट्सकोडमधील दोष शोधणे/ (Find Defects in Documents/Code, Logic)	सॉफ्टवेअरच्या वर्तनाची पडताळणी (Validate Software Behaviour)
तंत्रे (Techniques)	रिव्ह्यूज)Reviews), इन्स्पेक्शन्स) Inspections), स्टॅटिक ॲनालिसिस) Static Analysis)	युनिट, इंटिग्रेशन, सिस्टिम, ॲक्सेप्टन्स टेस्टिंग (Unit, Integration, System, Acceptance Testing)
खर्च (Cost)	कमी (Lower — Early Detection)	जास्त (Higher — Requires Test Environment)
सापडलेले दोष (Defects Found)	सिंटॅक्स, डिझाईन एरर्स (Syntax, Design Errors)	रनटाइम, लॉजिकल एरर्स (Runtime, Logical Errors)

3.6.4. चाचणीची पातळी (Levels of Testing)

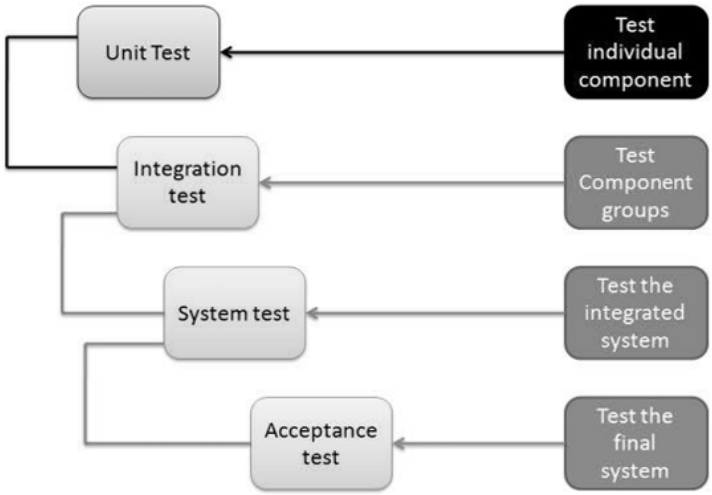


Fig. 3.42: चाचणीची पातळी (Levels of Testing)

3.6.4.1 युनिट टेस्टिंग(Unit Testing)

युनिट म्हणजे सॉफ्टवेअर सिस्टिममधील सर्वात लहान टेस्ट करण्यायोग्य घटक (Smallest Testable Part of the Software System). युनिट टेस्टिंग हे तपासण्यासाठी केले जाते की सॉफ्टवेअरमधील सर्वात कमी पातळीवरील स्वतंत्र घटक (Independent Entities) योग्य प्रकारे कार्य करतात का. सर्वात कमी स्तरावर होणाऱ्या टेस्टिंगला युनिट टेस्टिंग किंवा मॉड्यूल टेस्टिंग (Module Testing) असे म्हणतात .निट्स टेस्ट केल्यानंतर आणि लोलेव्हल बग्स- (Low-Level Bugs) शोधून दुरुस्त केल्यानंतर, त्या युनिट्स एकत्र करून)Integrated) इंटिग्रेशन टेस्टिंग (Integration Testing) केली जाते .जेव्हा डेव्हलपर सॉफ्टवेअर कोडिंग करत असतो, तेव्हा कधी कधी डिपेंडंट मॉड्यूल्स (Dependent Modules) पूर्ण झालेले नसतात. अशा परिस्थितीत डेव्हलपर Stub आणि Driver वापरतो

- Stub: Called Unit चे सिम्युलेशन करतो.
- Driver: Caller Unit चे सिम्युलेशन करतो.

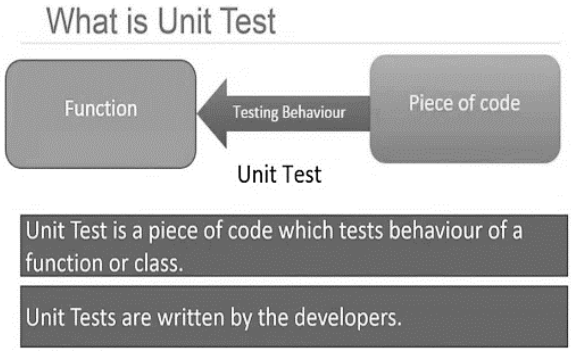


Fig. 3.43. युनिट टेस्टिंग (Unit Testing)

युनिट टेस्टिंगसाठी स्टब आणि ड्रायव्हर्सची ट्रान्सलेशन असते, स्टब नावाच्या युनिटचे अनुकरण करतात आणि ड्रायव्हर कॉलिंग युनिटचे अनुकरण करतात.

ड्रायव्हर (Driver)

- Driver हा एक सोप्या प्रकारचा Main प्रोग्राम(Program) (Simple Main Program) असतो.
- तो Test Case Data स्वीकारतो (Accepts Test Case Data).
- हे डेटा(Data) Test होणाऱ्या घटकाला)Component) पाठवतो.
- Returned Results प्रिंट करतो .दर्शवतो /

स्टब(Stub)

- Stub हे असे Modules Replace करतात, जे Component द्वारे कॉल केले जातात (Modules that are Called by the Component Under Test).
- Stub सामान्यतः:
  - किमान डेटाप्रोसेसिंग (Data) (Minimal Data Manipulation) करते.
  - Entry ची पडताळणी (Verification of Entry) पुरवते.
  - आणि नंतर Control पुन्हा Testing Component कडे परत करते.

ड्रायव्हर आणि स्टब दोघेही ओव्हरहेडचे प्रतिनिधित्व करतात

दोन्ही लिहिणे आवश्यक आहे परंतु सॉफ्टवेअर उत्पादनाचा भाग बनवू नका

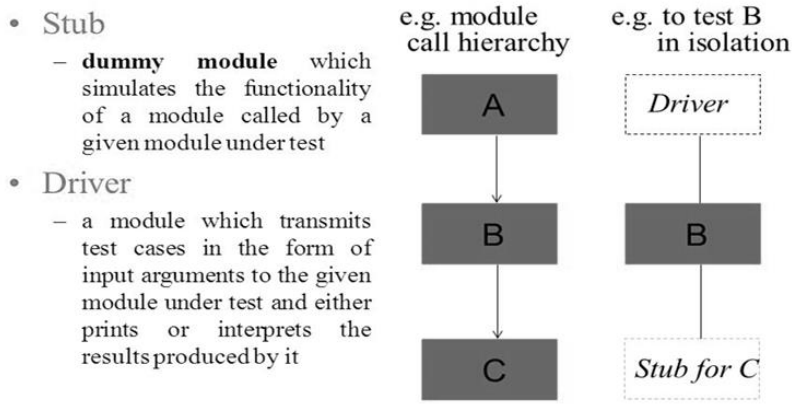


Fig. 3.44: स्टड आणि ड्रायव्हर (Stud and Driver)

3.6.4.2 इंटिग्रेशन टेस्टिंग (Integration Testing)

जेव्हा युनिट्स (Units) टेस्ट होतात आणि लोलेव्हल बग्स- (Low-Level Bugs) शोधून दुरुस्त केले जातात, तेव्हा हे युनिट्स एकत्र)Integrated) केले जातात .यानंतर इंटिग्रेशन टेस्टिंग (Integration Testing) Modules च्या समूहावर (Groups of Modules) केली जाते .हा Incremental Testing चा Process असतो — जसजसे अधिकाधिक सॉफ्टवेअरचे भाग एकत्र जोडले जातात, तसतसे अधिक Testing होते .अखेरीस संपूर्ण सॉफ्टवेअर प्रॉडक्ट किंवा त्याचा एक मोठा भाग System Testing मध्ये एकाच वेळी टेस्ट केला जातो.

3.6.4.2 इंटिग्रेशन टेस्टिंगसाठी दृष्टिकोन (Approaches to Integration Testing)

3.6.4.2.1 टॉप डाउन टेस्टिंग (Top-down Testing)

या पद्धतीमध्ये Main Module पासून Sub Modules कडे Testing केली जाते. जर एखादा Sub Module अजून विकसित)Developed) झालेला नसेल, तर त्या Sub Module साठी Stub नावाचा Temporary Program वापरला जातो .Stub Sub Module चं सिम्युलेशन करतो (Simulate the Sub Module), ज्यामुळे Main Module Testing योग्यरित्या करता येते.

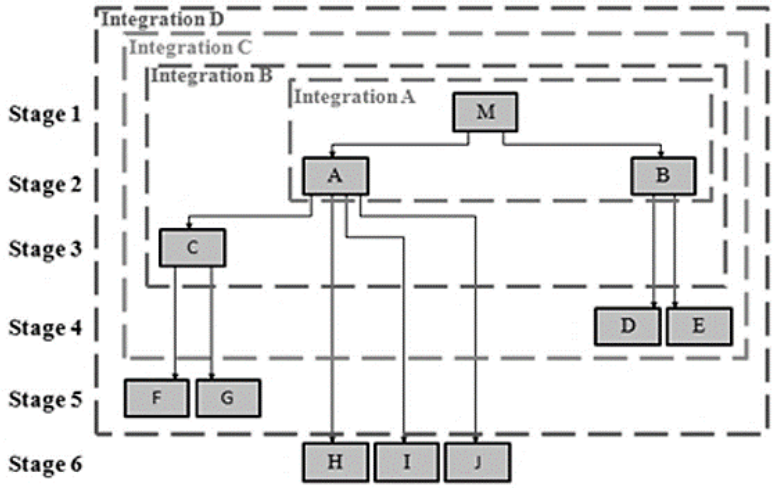


Fig. 3.45: टॉप-डाऊन टेस्टिंग (Top-down testing)

3.6.4.2.2. बॉटम अप टेस्टिंग (Bottom-up Testing)

- या पद्धतीमध्ये Sub Module पासून Main Module कडे Testing केली जाते.
- जर Main Module अजून विकसित)Developed) झालेला नसेल, तर त्या Main Module साठी Driver नावाचा Temporary Program वापरला जातो.
- Driver Main Module चं सिम्युलेशन करतो (Simulate the Main Module), ज्यामुळे Sub Modules चं Testing योग्यरित्या करता येते.

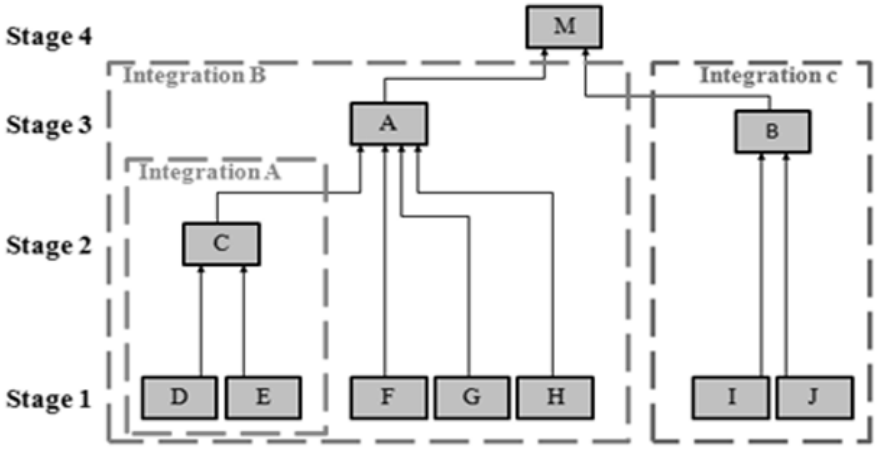


Fig. 3.46: बॉटम-अप चाचणी (Bottom-up testing)

3.6.4.3. सिस्टिम टेस्टिंग (System Testing)

युनिट्स(Units) एकत्र करून इंटिग्रेशन टेस्टिंग (Integration Testing) Modules च्या समूहावर)Groups of Modules) केली जाते. हा Incremental Testing चा Process असतो ज्यात अधिकाधिक सॉफ्टवेअरचे भाग)Pieces of Software) एकत्र जोडले जातात. हा Process सुरू राहतो जोपर्यंत पूर्ण सॉफ्टवेअर प्रॉडक्ट किंवा त्याचा मोठा भाग एकत्र येऊन एकाचवेळी टेस्ट केला जातो ह्यालाच System Testing म्हणतात.

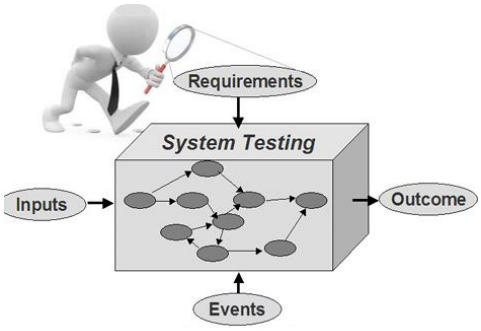


Fig. 3.47: सिस्टम टेस्टिंग (System Testing)

सिस्टम टेस्टिंग ही एक ब्लॅकबॉक्स टेस्टिंग टेक्निक- आहे जी संपूर्ण सिस्टम(System) निर्दिष्ट केलेल्या रिक्वायरमेंट्सच्या विरुद्ध तपासण्यासाठी वापरली जाते .System Testing मध्ये सिस्टमच्या फंक्शनॅलिटीज (Functionalities) End-to-End दृष्टिकोनातून तपासल्या जातात .यामध्ये अनेक सॉफ्टवेअर आणि हार्डवेअर कंपॅटिबिलिटी इश्यूज देखील उघड करता येतात System Test Execution दरम्यान असे इश्यूज ओळखले जातात .System Testing मध्ये दोन्ही प्रकारच्या टेस्टिंगचा समावेश असतो: फंक्शनल टेस्टिंग (Functional Testing), नॉनफंक्शनल टेस्टिंग- (Non-Functional Testing)

3.6.4.4 ॲक्सेप्टन्स टेस्टिंग)Acceptance Testing)

ॲक्सेप्टन्स टेस्टिंग चा उद्देश म्हणजे युजर सॅटिस्फॅक्शन (User Satisfaction) पडताळणे. ही टेस्टिंग रिअलवर्ल्ड - सीनारिओज (Real-World Scenarios) च्या आधारे केली जाते .

उदा:. विद्यार्थी प्रवेश)Student Enrolment) वर्कफ्लो युनिव्हर्सिटी सॉफ्टवेअरमध्ये योग्यरित्या कार्य करतो का हे तपासणे. Acceptance Testing द्वारे हे सुनिश्चित केले जाते की अंतिम सॉफ्टवेअर प्रॉडक्ट हे युजरच्या अपेक्षा)User Expectations) आणि व्यावसायिक गरजा)Business Needs) पूर्ण करते.

3.3.5 व्ही मॉडेल (V-Model) in SDLC

व्ही) (Model)मॉडेल-V-Model) हे सॉफ्टवेअर डेव्हलपमेंट लाईफ सायकल)SDLC) मधील एक महत्त्वाचे मॉडेल(Model) आहे, जे डेव्हलपमेंट आणि टेस्टिंग फेजेस यांना सिंक्रोनाइझ (Synchronize) करतेया . मध्ये(Model)मॉडेल प्रत्येक डिझाईन स्टेजसाठी (Each Design Stage) एक संबंधित व्हॅलिडेशन स्टेप)Corresponding Validation Step) असतेयामुळे. सॉफ्टवेअरच्या प्रत्येक टप्प्यावर पडताळणी)Validation) होते आणि त्रुटी लवकरच शोधल्या जातात.

Phases (टप्पे:(

1. **व्हेरिफिकेशन (Verification)— डाव्या बाजूचा भाग (Left Arm):**
   - रिक्वायरमेंट्स ॲनालिसिस)Requirements Analysis): ॲक्सेप्टन्स टेस्ट प्लॅनिंग (Acceptance Test Planning) शी संबंधित असते.
   - सिस्टम) डिझाईन (System)System Design): सिस्टमटेस्ट केसेस (System) (System Test Cases) शी संरेखित केले जाते — उदा :.IoT आर्किटेक्चर्ससाठी.
2. **व्हॅलिडेशन (Validation)— उजव्या बाजूचा भाग (Right Arm)**
   - युनिट टेस्टिंग)Unit Testing): कोड मॉड्यूल्सची पडताळणी (Validates Code Modules) Detailed Design Specs वरून.
   - इंटिग्रेशन टेस्टिंग)Integration Testing): कंपोनंट इंटरॲक्शन (Component Interactions) आर्किटेक्चरल डिझाईनशी जुळतात का हे सुनिश्चित करते.

फायदे (Advantages)

1. डिफेक्ट्स लवकर शोधल्यामुळे (Early Defect Detection)- ऑटोमोटिव्ह सॉफ्टवेअर प्रोजेक्ट्स मध्ये रीवर्कचा खर्च)Rework Costs) कमी होतो.

2. रिक्वायरमेंट्स आणि टेस्ट केसेस यांच्यात सुस्पष्ट ट्रेसिबिलिटी)Clear Traceability)- ऍजाईल एन्व्हायरमेंट्स (Agile Environments) मध्ये मिळते.

References:

1. Software Engineering – A Practitioner's Approach, 7th Edition, Roger Pressman

Websites:

1. Swayam NPTEL 12 Week Course on Software Testing by Dr. T. Subha https://onlinecourses.swayam2.ac.in/ntr25_ed118/preview
2. Data Flow Diagram - https://www.geeksforgeeks.org/software-engineering/what-is-dfddata flow-diagram/
3. Software Testing - https://www.guru99.com/software-testing.html
4. Types of Software Testing - https://www.softwaretestinghelp.com/types-of-software-testing/

युनिट- 4
सॉफ्टवेअर प्रोजेक्ट कॉस्ट एस्टिमेशन
(Software Project Cost Estimation)

विषय निष्पत्ती (Course Outcome):

CO4 : सॉफ्टवेअर उत्पादनासाठी वेगवेगळ्या नियोजन आणि खर्च अंदाज तंत्रांचा वापर करणे.

घटक निष्पत्ती (Theory Learning Outcome - TLO):

1. सॉफ्टवेअरप्रोजेक्टसाठीव्यवस्थापनस्पेक्ट्रमस्पष्टकरणे.
2. सॉफ्टवेअर उत्पादनाचा आकार अंदाजे काढा.
3. अनुभवजन्य पद्धतीचा वापर करून सॉफ्टवेअर उत्पादनाची अंदाजितकिंमतकरणे.
4. COCOMO मॉडेल वापरून दिलेल्या सॉफ्टवेअरचा आकार मोजा.
5. कोणत्याही सॉफ्टवेअर डेव्हलपमेंट समस्येसाठी ओळखल्या जाणाऱ्या जोखमींमध्ये RMMM रणनीती लागू करणे.

4.1 व्यवस्थापन स्पेक्ट्रम - 4पी.

व्यवस्थापन स्पेक्ट्रम - 4पी. हे प्रोजेक्ट व्यवस्थापनात, विशेषतः सॉफ्टवेअर अभियांत्रिकीमध्ये, प्रोजेक्टच्या यशावर परिणाम करणारे चार प्रमुख घटक दर्शविणारे एक मॉडेल आहे. हे आहेत:

1. लोक (People)

- कोणत्याही प्रोजेक्टतील सर्वात महत्त्वाचा घटक aahe.
- प्रोजेक्ट व्यवस्थापक, विकासक, परीक्षक, क्लायंट आणि भागधारक यांचा समावेश आहे.
- महत्वाचे असणे यावर भर देतो:
    - o संवाद कौशल्ये
    - o नेतृत्व आणि प्रेरणा
    - o भूमिका आणि जबाबदाऱ्या
    - o संघ बांधणी आणि गतिशील असणे

2. उत्पादन (Product)

- विकसित किंवा वितरित केले जात असलेल्या गोष्टींचा संदर्भ देते.
- सॉफ्टवेअर, सिस्टम, सेवा किंवा इतर कोणत्याही वितरित करण्यायोग्य गोष्टी असू शकतात.
- समाविष्ट आहे:
    - o आवश्यक विश्लेषण
    - o उत्पादन उद्दिष्टे
    - o व्याप्ती व्याख्या
    - o कार्यात्मक आणि अकार्यक्षम तपशील

3. प्रोसेस (Process)

- प्रोजेक्ट व्यवस्थापित करण्यासाठी आणि अंमलात आणण्यासाठी वापरल्या जाणाऱ्या चौकटी किंवा पद्धतीची व्याख्या करते.
- उदाहरणे:
    - o वॉटरफॉल
    - o एजाईल (स्क्रम, कॅनबॅन)
    - o स्पायरल, व्ही-मॉडेल, डेव्हऑप्स, इ.
- हे सुनिश्चित करण्यास मदत करते:
    - o सुसंगतता
    - o गुणवत्ता नियंत्रण
    - o अंदाजे परिणाम

4. प्रोजेक्ट (Project)

- उत्पादन वितरित करण्यात गुंतलेल्या प्रत्यक्ष क्रियाकलाप, नियोजन आणि अंमलबजावणी.
- लक्ष केंद्रित क्षेत्रे:
    - o नियोजन आणि वेळापत्रक
    - o जोखीम व्यवस्थापन
    - o संसाधन वाटप
    - o देखरेख आणि नियंत्रण
    - o बजेट आणि वेळ व्यवस्थापन

4.2 आकार अंदाजासाठी चे मापन (Metrics for Size Estimation): कोडची ओळ (LoC), फंक्शन पॉइंट्स (FP).

4.2.1 आकार अंदाजासाठी मेट्रिक्स: कोडची ओळ (LoC)

सॉफ्टवेअर अभियांत्रिकीमध्ये कोडची ओळ (LoC) ही सर्वात मूलभूत आणि मोठ्या प्रमाणावर वापरली जाणारी आकार अंदाज मेट्रिक्सपैकी एक आहे. ती सॉफ्टवेअर प्रोजेक्टचा आकार, प्रयत्न आणि त्यांची उत्पादकतेचा अंदाज लावण्यास मदत करतात.

4.2.1.1 कोडची ओळ (LoC) म्हणजे काय?

कोडची ओळ सॉफ्टवेअर प्रोग्रामच्या सोर्स कोड मधील ओळींच्या संख्या दर्शवते. यामध्ये एक्झिक्युटेबल कोड लाईन्स समाविष्ट असतात आणि कधीकधी प्रकारानुसार टिप्पण्या आणि रिकाम्या ओळी वगळल्या जातात (खाली दर्शिवल्या प्रमाणेपहा).

4.2.1.2 एल ओ सी मेट्रिक्सचे प्रकार:

प्रकार	वर्णन
भौतिक LoC	ही सोर्स फाइलमधील प्रत्येक ओळ मोजते, ज्यामध्ये टिप्पण्या आणि रिकाम्या ओळींचा समावेश असतो
तार्किक LoC	यात फक्त एक्झिक्युटेबल स्टेटमेंट असलेल्या ओळी ची संख्या मोजते.
टिप्पणी LoC	यात फक्त टिप्पण्या असलेल्या ओळी मोजते.
स्रोत LoC (SLOC)	टिप्पण्या आणि रिकाम्या ओळी वगळून प्रत्यक्ष सोर्स कोड मोजते.

4.2.1.3 LoC चे उपयोग (Uses):

- प्रयत्नांचा अंदाज: विकासकाच्या प्रयत्नांचा आणि प्रोजेक्टच्या कालावधीचा अंदाज घेण्यास मदत करते.
- उत्पादकता मापन: उत्पादनाचे मूल्यांकन करण्यासाठी वापरले जाते, उदा., प्रति व्यक्ती दरमहा LoC.
- खर्चाचा अंदाज: जास्त LoC विकास आणि देखभाल खर्च जास्त दर्शवू शकते.
- गुणवत्ता मूल्यांकन: खूप जास्त LoC जटिलता किंवा फुगलेला कोड सूचित करू शकते.

4.2.1.4 LoC च्या मर्यादा (Limitations):

- भाषा अवलंबून: LoC वेगवेगळ्या भाषांमध्ये लक्षणीयरीत्या बदलते (Python vs Java).
- गुणवत्ता मोजत नाही: अधिक कोड म्हणजे चांगली कार्यक्षमता नाही.
- दिशाभूल करणारे असू शकते: लॉजिकमध्ये थोडासा बदल LoC मध्ये फारसा प्रतिबिंबित होऊ शकत नाही.

4.2.1.5 सर्वोत्तम पद्धती (Best Practices):

- फंक्शन पॉइंट्स (FP), सायक्लोमॅटिक कॉम्प्लेक्सिटी इत्यादी इतर मेट्रिक्ससह LoC वापरणे.
- लॉजिकल LoC ला प्राधान्य अधिक अचूक आणि भाषा-स्वतंत्र विश्लेषण करता येते
- LoC मोजणी स्वयंचलित करणे करण्यासाठी CLOC, SLOC Count किंवा IDE प्लगइन्स सारख्या साधनांचा वापर करवा.

4.2.2 मेट्रिक्स: फंक्शन पॉइंट्स (FP).

फंक्शन पॉइंट्स (FP) अंदाजित आकाराचे मापन: फंक्शन पॉईंट अनालिसिस हे एक टेक्निक असून ते सॉफ्टवेअरची फंक्शनल sizeचे मोजमाप करण्यासाठी वापरले जाते यामध्ये वापर कर्त्याच्या दृष्टिकोनाचा विचार करून अंदाज लावला जातो यामध्ये किती लाईन आहेत याचा विचार नकरता युजरच्या दृष्टिकोनाच्या विचारावर अवलंबूनअसतो

4.2.2.1 फंक्शन पॉइंट्स कामहत्त्वाचेआहे

लाइन ऑफ कोड (LoC) च्या विपरीत, जे प्रोग्रामिंग भाषा आणि कोड शैलीवर अवलंबून असते, FP सॉफ्टवेअर काय करते यावर लक्ष केंद्रित करते - ते भाषेपासून स्वतंत्र बनवते आणि प्रोजेक्ट नियोजनाच्या सुरुवातीच्या टप्प्यात अधिक उपयुक्त बनवते.

4.2.2.2 FP चे प्रमुख घटक:

फंक्शन पॉइंट्स सिस्टमच्या 5 प्रमुख घटकांचे मूल्यांकन करून मोजले जातात:

घटक	वर्णन
बाह्य इनपुट (EI)	अंतर्गत डेटा अपडेट करणारे वापरकर्ता इनपुट (उदा., फॉर्म, व्यवहार).
बाह्य आउटपुट (EO)	वापरकर्त्याला पाठवलेला डेटा (उदा., अहवाल, संदेश).
बाह्य चौकशी (EQ)	वापरकर्त्याला पाठवलेला डेटा (उदा., अहवाल, संदेश).
अंतर्गत लॉजिकल फाइल्स (ILF)	सिस्टममध्ये साठवलेल्या डेटाचे वापरकर्ता-ओळखण्यायोग्य गट (उदा., टेबल्स)
बाह्य इंटरफेस फाइल्स (EIF)	अंतर्गत अपडेटशिवाय इनपुट/आउटपुट संयोजन (उदा., शोध क्वेरी).

प्रत्येक घटकाला त्याच्या जटिलतेनुसार कमी, सरासरी किंवा उच्च असे रेटिंग दिले जाते आणि वजन दिले जाते.

4.2.2.3 फंक्शन पॉइंट कॅल्क्युलेशन (मूलभूत पायऱ्या):

1. पाचही घटक ओळखून त्यांच्या वर्गीकरण करणे.
2. जटिलतेचे स्तर नियुक्त करून मानक वजने लागू करणे
3. अनअॅडजस्टेड फंक्शन पॉइंट्स (UFP) ची गणना करणे:
   UFP=∑ (घटक संख्या×जटिलता वजन) UFP = \बेरीज (घटक\ संख्या ×जटिलता\ वजन) UFP=∑ (घटक संख्या×जटिलता वजन)
4. 14 सामान्य प्रणाली वैशिष्ट्यांवर आधारित (उदा., कामगिरी, पोर्टेबिलिटी) मूल्य समायोजन घटक (VAF) मोजा.

FP=UFP×(0.65+0.01×TotalVAF) FP = UFP × (0.65 + 0.01 × एकूण VAF) FP=UFP×(0.65+0.01×TotalVAF).

4.2.2.4 FP चे फायदे (Advantages):

1. स्वतंत्रभाषा आणि तंत्रज्ञानअसते.
2. सुरुवाती पासून विकासाच्या जीवनचक्रात उपयुक्त असते
3. खर्च, प्रयत्न आणि उत्पादकता यांच्याअंदाजासाठी चांगले असते.
4. वापरकर्तकेंद्रित विकासाला प्रोत्साहन देते.

4.2.2.5 FP च्या मर्यादा:

1. प्रशिक्षित विश्लेषकांना सातत्याने अर्ज करण्याची आवश्यकता आहे.
2. जटिलता आणि वजनांचे मूल्यांकन करण्यात व्यक्तिनिष्ठता येते.
3. गैर-कार्यक्षम आवश्यकता थेट मोजत नाही (उदा. कामगिरी).

4.2.3 FP विरुद्ध LoC - जलद तुलना:

निकष (Criteria)	फंक्शन पॉइंट्स (Function Points)	लाईन्स ऑफ कोड (Lines of Code (LoC))
भाषेवर अवलंबून (Language Dependent)	नाही	हो
मापन केव्हा (Measured When)	सुरुवातीचा (आवश्यकता टप्पा)	उशिरा (कोडिंगनंतर)
लक्ष केंद्रित (Focus)	कार्यक्षमता	कोडआकार
अचूकता (Accuracy)	उच्च-स्तरीयअंदाजासाठीचांगले	सुरुवातीलाकमीअचूक

4.3 प्रोजेक्ट खर्च अंदाज पद्धती: ह्युरिस्टिक, विश्लेषणात्मक आणि अनुभवजन्य अंदाजाचा आढावा.

4.3.1 प्रोजेक्ट खर्च अंदाज पद्धती: ह्युरिस्टिकचा आढावा.

प्रोजेक्ट खर्च अंदाज पद्धतीमध्ये अनुभव-आधारित तंत्रे किंवा नियम-अनुरूप पद्धतींचा वापर करून जलद आणि वाजवी अचूक अंदाज लावणे समाविष्ट आहे. ते तपशीलवार गणितीय मॉडेलिंगऐवजी भूतकाळातील डेटा, तज्ञांचा निर्णय आणि अनुभवजन्य संबंधांवर आधारित आहे. ह्युरिस्टिक दृष्टिकोनात, अंदाजकर्ते बहुतेकदा मागील प्रोजेक्टमधून मिळवलेल्या सरलीकृत सूत्रे आणि उपमांवर अवलंबून असतात. टीम कौशल्य, तंत्रज्ञान स्टॅक, प्रोजेक्ट जटिलता आणि जोखीम पातळी यासारख्या घटकांचा वापर करून हे अंदाज समायोजित केले जाऊ शकतात. उदाहरणार्थ, जर 10000 ओळींच्या कोडच्या मागील प्रोजेक्टला 4 विकासकांच्या टीमसह ५ महिने लागले, तर समान जटिलता आणि आकार असलेल्या नवीन प्रोजेक्टचा अंदाज प्रमाणानुसार केला जाऊ शकतो. या पद्धतीची एक ताकद म्हणजे ती जलद निर्णय घेण्यास अनुमती देते, विशेषतः अशा परिस्थितीत जिथे तपशीलवार विश्लेषणासाठी वेळ आणि संसाधने मर्यादित असतात. प्रस्ताव विनंती (RFP) प्रतिसाद, बजेट नियोजन किंवा प्रारंभिक बोली प्रोसेस दरम्यान हे विशेषतः उपयुक्त आहे.

4.3.1.1 ह्युरिस्टिक दृष्टिकोनाची प्रमुख वैशिष्ट्ये (Key Features of the Heuristic Approach):

वैशिष्ट्य	वर्णन
अनुभवावर आधारित	तत्सम मागील प्रोजेक्टमधून मिळालेल्या कौशल्यांवर आणि धड्यांवर अवलंबून असतो.
नियमांवर आधारित	किंमत = a × (आकार)^b, जिथे a आणि b स्थिरांक आहेत अशी सूत्रे लागू करते.
जलद आणि व्यावहारिक	तपशीलवार डेटा उपलब्ध नसताना जलद अंदाज प्रदान करते.
लवचिक	व्याप्ती किंवा गृहीतकांमधील बदलांशी सहजपणे जुळवून घेता येते.

4.3.1.2 सामान्य ह्युरिस्टिक मॉडेल्स:

1. **COCOMO (रचनात्मक खर्च मॉडेल) – ह्युरिस्टिक मॉडेलचे एक उत्कृष्ट उदाहरण जे वापरते:**
   प्रयत्न (व्यक्ती-महिने) =a×(KLoC)b\text {प्रयत्न (व्यक्ती-महिने)} = a \times (\text {KLoC}) ^bप्रयत्न (व्यक्ती-महिने) =a×(KLoC)b
   जिथे a आणि b प्रोजेक्ट प्रकारावर अवलंबून असतात (सेंद्रिय, अर्ध-पृथक, एम्बेडेड organic, semi-detached, embedded).

2. **पुटनम मॉडेल (SLIM)** – प्रयत्न आणि वेळापत्रकाचा अंदाज घेण्यासाठी ऐतिहासिक उत्पादकता आणि कर्मचारी डेटा वापरते.
3. **तज्ञांचा निर्णय** - वरिष्ठ विकासक किंवा प्रोजेक्ट व्यवस्थापकांच्या अंतर्ज्ञान आणि अनुभवावर आधारित अंदाज.

4.3.1.3 फायदे (Advantages):

1. प्रोजेक्टच्या सुरुवातीच्या टप्प्यात जलद आणि वापरण्यास सोपे आहे.
2. तपशीलवार तपशील उपलब्ध नसताना उपयुक्त आहे.
3. संस्थात्मक ज्ञान आणि भूतकाळातील डेटाचा वापर करते.

4.3.1.4 मर्यादा (Limitations) :

1. ऐतिहासिक डेटा जुना किंवा असंबद्ध असल्यास अचूकतेचा अभाव असू शकतो.
2. व्यक्तिनिष्ठ — अंदाजकर्त्याच्या अनुभवावर मोठ्या प्रमाणात अवलंबून asto.
3. कोणताही पूर्वग्रह नसलेल्या जटिल किंवा नवीन प्रोजेक्टसाठी आदर्श नाही.

4.3.1.5 जिथे ह्युरिस्टिक अंदाज सर्वोत्तम काम करतो (Where Heuristic Estimation Works Best):

- लवकर व्यवहार्यता अभ्यास.
- पूर्वी पूर्ण झालेल्या कामांसारखे प्रोजेक्ट
- चांगल्या प्रकारे दस्तऐवजीकरण केलेल्या प्रोजेक्ट इतिहासासह संस्था(ज्या संस्थांचे मागील प्रोजेक्टत चांगले प्रकारे दस्तऐवजीकरण केलेले असते)

4.3.2 प्रोजेक्ट खर्च अंदाज पद्धती: विश्लेषणात्मक.

विश्लेषणात्मक दृष्टिकोन (ज्याला bottom-up estimation तळाशी-अप अंदाज असेही म्हणतात) ही प्रोजेक्ट खर्च अंदाजित करण्याची एक अत्यंत तपशीलवार पद्धत आहे. यामध्ये प्रोजेक्टचे लहान घटकांमध्ये (कार्ये किंवा कामाचे पॅकेजेस) विभाजन करणे आणि प्रत्येक घटकाची किंमत अंदाजित करणे, नंतर एकूण प्रोजेक्ट खर्च निश्चित करण्यासाठी त्यांना एकत्रित करणे समाविष्ट आहे.

4.3.2.1 प्रमुख वैशिष्ट्ये:

- **ग्रॅन्युलॅरिटी:** हे अंदाज कार्य किंवा क्रियाकलाप पातळीवर केले जातात.
- **अचूकता:** उच्च अचूकता देते, विशेषतः चांगल्या प्रकारे परिभाषित प्रोजेक्टसाठी.
- **प्रयत्न:** वेळखाऊ आणि संसाधन-केंद्रित aste.
- **आधार:** प्रत्येक घटकासाठी ऐतिहासिक डेटा, तज्ञांचा निर्णय किंवा विक्रेत्याच्या कोट्सवर अवलंबून असतो.

4.3.2.2 समाविष्टअसलेल्यापायऱ्या

1. **कामाचे विभाजन संरचना (WBS):** प्रोजेक्टचे व्यवस्थापनीय घटकांमध्ये विघटन करणे.
2. **संसाधनांचा अंदाज घ्या:** प्रत्येक कामासाठी आवश्यक असलेले कामगार, साहित्य, उपकरणे आणि इतर संसाधने ओळखा.
3. **युनिट खर्च निश्चित करणे:** प्रति युनिट खर्च निश्चित करण्यासाठी ऐतिहासिक डेटा किंवा बाजार दर वापरा.
4. **कार्य खर्चाची गणना करणे:** आवश्यक प्रमाणात युनिट खर्च गुणाकार करणे.
5. **एकूण खर्च:** एकूण प्रोजेक्ट खर्च मिळविण्यासाठी सर्व वैयक्तिक कार्य खर्चाची बेरीज करणे.
6. **आकस्मिकता समाविष्ट करणे:** अनिश्चितता किंवा जोखीम लक्षात घेण्यासाठी बफर जोडा.

4.3.2.3 फायदे (Advantages):

1. उच्च अचूकता आणि तपशील.
2. तपशीलवार बजेटिंग आणि खर्च नियंत्रणास समर्थन देते.
3. प्रत्येक खर्च आयटमसाठी ट्रेसेबिलिटी प्रदान करते.

4.3.2.4 तोटे (Disadvantages):

1. लक्षणीय वेळ आणि कौशल्याचीआवश्यकताअसते.
2. मर्यादित माहितीसह प्रोजेक्टच्या सुरुवातीच्या टप्प्यांसाठी आदर्श नाही.

3. अनेक कामांसह मोठ्या प्रोजेक्टसाठी ते जटिल होऊ शकते.

4.3.2.5 कधी वापरावे

- सुस्पष्ट व्याप्ती आणि आवश्यकता असलेले प्रोजेक्ट.
- बांधकाम, उत्पादन किंवा अभियांत्रिकी प्रोजेक्ट.
- जेव्हा उच्च किमतीची अचूकता महत्त्वाची असते (उदा., सरकारी किंवा निश्चित-किंमत करणेर).

4.3.2.6 उदाहरण

वेबसाइट तयार करण्याची कल्पना करणे:

• **कार्य:** होमपेज डिझाइन करणे
  - o तास: 20
  - o खर्च/तास: $50
  - o खर्च: 20 × $50 = $1000

• **कार्य:** बॅकएंड इंटिग्रेशन
  - o तास: 40
  - o खर्च/तास: $60
  - o खर्च: 40 × $60 = $2400

एकूण अंदाजे खर्च = $1000 + $2400 + ... (इतर कामे)

4.3.3 प्रोजेक्ट खर्च अंदाज पद्धती: अनुभवजन्य अंदाज.

प्रोजेक्ट खर्च अंदाज करण्यासाठी अनुभवजन्य अंदाज पद्धत ऐतिहासिक डेटा, अनुभव आणि गणितीय मॉडेल्सवर आधारित आहे. ती तपशीलवार कार्य विश्लेषणा ऐवजी वास्तविक-जगातील प्रोजेक्ट डेटावरून मिळवलेल्या सूत्रे आणि अल्गोरिदम वर अवलंबून असते.

4.3.3.1 प्रमुख वैशिष्ट्ये

1. डेटा-चालित: खर्च अंदाज करण्यासाठी मागील प्रोजेक्टमधील डेटा वापरते.
2. मॉडेल-आधारित: अनेकदा COCOMO किंवा फंक्शन पॉइंट विश्लेषण सारख्या खर्च अंदाज मॉडेल्सचा वापर करते.
3. सरलीकृत इनपुट: आकार (उदा. कोडच्या ओळी, फंक्शन पॉइंट्स) आणि जटिलता यासारख्या उच्च-स्तरीय इनपुटची आवश्यकता असते.
4. पुनरावृत्ती करण्यायोग्य: समान वैशिष्ट्यांसह समान प्रोजेक्टमध्ये पुन्हा वापरता येते.

4.3.3.2 सामान्य अनुभवजन्य मॉडेल

1. COCOMO (रचनात्मक खर्च मॉडेल)

- o कोडच्या ओळी आणि प्रोजेक्टच्या जटिलतेवर आधारित खर्चाचा अंदाज लावतो.
- o उदाहरण सूत्र:

  प्रयत्न = a × (आकार)^b × EAF

  (जिथे a आणि b स्थिरांक आहेत, आकार = KLOC, EAF = प्रयत्न समायोजन घटक)

2. फंक्शन पॉइंट विश्लेषण (FPA)

- o वापरकर्त्याला दिलेली कार्यक्षमता मोजते.
- o LOC पेक्षा अधिक अमूर्त आणि व्यावसायिक अनुप्रयोगांसाठी योग्य.

3. SLIM, SEER-SEM, Putnam मॉडेल

- o गणितीय अंदाज तंत्रांवर अवलंबून असलेले इतर अनुभवजन्य मॉडेल.

4.3.3.3 फायदे (Advantages):

1. मॉडेल्स कॅलिब्रेट केल्यानंतर जलद आणि कार्यक्षम होतात.
2. तपशीलवार माहिती उपलब्ध नसताना प्रोजेक्टच्या सुरुवातीच्या टप्प्यात उपयुक्त.

3. प्रोजेक्टमध्ये मानकीकरणाला प्रोत्साहन देते.

4.3.3.4 तोटे (Disadvantages) :

1. अचूक ऐतिहासिक डेटा आणि मॉडेल कॅलिब्रेशन आवश्यक आहे.
2. अद्वितीय किंवा नाविन्यपूर्ण प्रोजेक्टसाठी कमी अचूकआहे.
3. मॉडेल निवड आणि इनपुट अचूकतेवर मोठ्या प्रमाणात अवलंबून असते.

4.3.3.5 कधी वापरावे (When to Use) :

- प्रोजेक्ट नियोजनाच्या सुरुवातीच्या टप्प्यातवापरावे.
- भूतकाळात पूर्ण झालेल्या प्रोजेक्टसारख्या प्रोजेक्टसाठीवापरातात.
- जेव्हा जलद, ढोबळ अंदाज स्वीकार्य असतात.

4.3.3.6 उदाहरण (COCOMO बेसिक मॉडेल वापरून)

- अंदाजे आकार = 50 किलो कॅलरीज
- प्रकार: सेंद्रिय प्रोजेक्ट
- सूत्र: प्रयत्न = 2.4 × (आकार)^1.05

  → प्रयत्न = 2.4 × (50) ^1.05 ≈ 2.4 × 56.27 ≈ 135 व्यक्ती-महिने.

4.4 COCOMO (रचनात्मक खर्च मॉडेल), COCOMO II

4.4.1 COCOMO (रचनात्मक खर्च मॉडेल)

1981 मध्ये बॅरी बोहम यांनी विकसित केलेले COCOMO मॉडेल हे सॉफ्टवेअर डेव्हलपमेंट प्रोजेक्टसाठी आवश्यक असलेल्या प्रयत्नांचा, खर्चाचा आणि वेळापत्रकाचा अंदाज घेण्यासाठी वापरले जाणारे एक अनुभवजन्य प्रोजेक्ट अंदाज मॉडेल आहे. ते प्रोजेक्ट पॅरामीटर्सचा अंदाज लावण्यासाठी ऐतिहासिक प्रोजेक्ट डेटा आणि गणितीय समीकरणांचा वापर करते.

4.4.1.1 मुख्य संकल्पना: COCOMO सॉफ्टवेअर प्रोजेक्टच्या आकारावर आधारित प्रयत्नांचा (व्यक्ती-महिने) अंदाज लावते (KLOC मध्ये मोजले जाते - कोडच्या हजारो ओळी) आणि प्रोजेक्ट, उत्पादन आणि कर्मचारी वैशिष्ट्यांसाठी समायोजन घटक लागू करते.

4.4.1.2 COCOMO मॉडेल्स:

COCOMO मॉडेलचे तीन स्तर आहेत:

1. मूलभूत COCOMO

- प्रयत्नांचा अंदाजे अंदाज प्रदान करते.
- **सूत्र:**

  प्रयत्न = a × (KLOC)^b

  वेळ = c × (प्रयत्न)^d
- प्रोजेक्टच्या प्रकारानुसार स्थिरांक (a, b, c, d) बदलतात.

2. मध्यवर्ती COCOMO

- 15 खर्चाचे चालक (उदा., जटिलता, विश्वासार्हता, संघ क्षमता) जोडते.
- **सूत्र:**

  प्रयत्न = a × (KLOC)^b × EAF

  जिथे EAF = प्रयत्न समायोजन घटक.

3. तपशीलवार COCOMO

- टप्प्याटप्प्याने ब्रेकडाउन आणि अधिक अचूक खर्चाचे चालकांसह मॉडेलला आणखी परिष्कृत करते.

4.4.1.3 COCOMO मधील प्रोजेक्ट श्रेणी

प्रकार	वर्णन	स्थिरांक (मूलभूत)
Organic	साधे, लहान संघ, परिचित क्षेत्रे	a = 2.4, b = 1.05
Semi-Detached	मध्यम आकाराचे, मिश्र संघ, मध्यम जटिल	a = 3.0, b = 1.12
Embedded	जटिल प्रणाली, कडक निर्बंध	a = 3.6, b = 1.20

4.4.1.4 उदाहरण (ऑरगॅनिक प्रोजेक्टसाठी बेसिक COCOMO)

समजा तुमच्याकडे 50 KLOC आकाराचे सॉफ्टवेअर आहे:

- प्रयत्न = 2.4 × (50) ^1.05 ≈ 135 व्यक्ती-महिने
- वेळ = 2.5 × (135) ^0.38 ≈ 14.8 महिने

4.4.1.5 फायदे (Advantages):

1. वापरण्यास आणि समजण्यास सोपे.
2. वास्तविक प्रोजेक्ट डेटावर आधारित.
3. लहान ते मोठ्या प्रोजेक्टमध्ये स्केलेबल.

4.4.1.6 तोटे (Disadvantages):

1. अचूक आकार अंदाज (KLOC) आवश्यक आहे.
2. आधुनिक ॲजाईल किंवा ऑब्जेक्ट-ओरिएंटेड प्रोजेक्टसाठी योग्य नसू शकते.
3. नवीन तंत्रज्ञान किंवा पद्धतींसाठी मर्यादित लवचिकताअसते.

4.4.2 COCOMO II

4.4.2.1 COCOMO II ही मूळ COCOMO मॉडेलची अद्ययावत आवृत्ती आहे जी बॅरी बोहम आणि यूएससी येथील इतरांनी विकसित केली आहे, ज्यामध्ये ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग, रीयूझ, स्पायरल आणि ॲजाईल मॉडेल्स, रॅपिड ॲप्लिकेशन डेव्हलपमेंट इत्यादी आधुनिक सॉफ्टवेअर डेव्हलपमेंट पद्धतींचा समावेश आहे. हे अधिक लवचिक, तपशीलवार आणि समकालीन सॉफ्टवेअर अभियांत्रिकी प्रोजेक्टसाठी योग्य आहे.

4.4.2.2 COCOMO II चा उद्देश:

अनुमान करू शकणारे वास्तववादी आणि लवचिक मॉडेल प्रदान करते

- प्रयत्न (व्यक्ती-महिने)
- वेळ (विकास वेळापत्रक)
- सॉफ्टवेअर प्रोजेक्टचीकिंमत

ते पुनरावृत्ती विकास, घटक-आधारित प्रणाली आणि पुनर्वापरास समर्थन देण्यासाठी तयार केले आहे.

4.4.2.3 COCOMO II मॉडेल घटक:

COCOMO II मध्ये तीन उप-मॉडेल आहेत, प्रत्येक सॉफ्टवेअर डेव्हलपमेंट लाइफ सायकलच्या वेगळ्या टप्प्यासाठी योग्य आहे

1. ॲप्लिकेशन कंपोझिशन मॉडेल
    - सुरुवातीच्या टप्प्यात वापरले जाते (उदा., प्रोटोटाइपिंग).
    - ऑब्जेक्ट पॉइंट्स (स्क्रीन, रिपोर्ट्स, 3GL मॉड्यूल्स) वर आधारित.
2. सुरुवातीचे डिझाइन मॉडेल
    - आर्किटेक्चर परिभाषित झाल्यानंतर वापरले जाते.
    - KSLOC मध्ये फंक्शन पॉइंट्स किंवा आकारावर आधारित खर्च ड्रायव्हर्स (7 ड्रायव्हर्स) आणि अंदाजांचा एक मोठा संच वापरला जातो.

3. पोस्ट-आर्किटेक्चर मॉडेल

- सिस्टम आर्किटेक्चर अंतिम झाल्यानंतर वापरले जाते.
- सर्वात तपशीलवार आणि व्यापकपणे वापरले जाणारे स्वरूप.
- उपयोग:
    - o स्केल घटक (5)
    - o प्रयत्न गुणक (17 खर्च ड्रायव्हर्स)

4.4.2.4 आर्किटेक्चर मॉडेल सूत्र (Formula)

प्रयत्न (PM)=A×(आकार)B×∏EMi\text {प्रयत्न (PM)} = A \times (\text{आकार}) ^{B} \times \prod EM_iEffort (PM)=A×(आकार)B×∏EMi

कुठे (Where):

- A = स्थिरांक (सहसा 2.94)
- आकार = KSLOC (कोडच्या हजारो स्त्रोत रेषा)
- B = 0.91 +0.01 × ∑ (स्केल घटक)
- EM_i = 17 खर्च चालकांमधील प्रयत्न गुणक

4.4.2.5 प्रमुख इनपुट (Key Inputs):

5 स्केल घटक (घातांक B ला प्रभावित करतात):

1. पूर्वस्थिती
2. विकास लवचिकता
3. आर्किटेक्चर/जोखीम निराकरण
4. टीम कोहेजन
5. प्रोसेस परिपक्वता

17 खर्च चालक (प्रयत्नांना प्रभावित करतात रेषीय), यामध्ये गटबद्ध केले आहे (Cost Drivers (affect effort linearly), grouped into):

- उत्पादन (उदा., जटिलता, आवश्यक विश्वासार्हता)
- प्लॅटफॉर्म (उदा., रन-टाईम कामगिरी, प्लॅटफॉर्म अस्थिरता)
- कर्मचारी (उदा., अनुभव, क्षमता)
- प्रोजेक्ट (उदा., साधने, वेळापत्रक, पुनर्वापर)

4.4.2.6 फायदे (Advantages):

- आधुनिक विकास पद्धती प्रतिबिंबित करते.
- पुनर्वापर आणि जलद विकासास समर्थन देते.
- सानुकूल करण्यायोग्य आणि स्केलेबल.

4.4.2.7 तोटे (Disadvantages):

- तपशीलवार इनपुट आवश्यक आहेत.
- सर्वोत्तम अचूकतेसाठी जटिल कॅलिब्रेशन आवश्यक आहे.
- अगदी सुरुवातीच्या प्रोजेक्ट टप्प्यांसाठी उपयुक्त नाही (ॲप्लिकेशन कंपोझिशन मॉडेलसह वापरल्याशिवाय).

4.4.2.8 उदाहरण (स्थापत्य शास्त्रानंतर) (post-architecture)

- आकार = 50 KSLOC
- A = 2.94
- B = 1.1
- ∏EM = 1.5 (निवडलेल्या खर्चाच्या चालकांचे उत्पादन)

प्रयत्न=2.94× (50)1.1×1.5≈2.94×66.7×1.5≈294.3 व्यक्ती-महिने.

4.5 जोखीम विश्लेषण आणि व्यवस्थापन: जोखीम ओळखणे, जोखीम मूल्यांकन, जोखीम व्यवस्थापन आणि देखरेख, जोखीम शुद्धीकरण आणि कमी करणे, RMMM योजना.

4.5.1 जोखीम विश्लेषण आणि व्यवस्थापन: जोखीम ओळखणे.

जोखीम ओळखणे ही जोखीम विश्लेषण आणि व्यवस्थापन प्रक्रियेतील पहिली आणि मूलभूत पायरी आहे. यामध्ये प्रोजेक्टच्या परिणामावर नकारात्मक किंवा सकारात्मक परिणाम करू शकणाऱ्या संभाव्य घटना किंवा परिस्थितींची पद्धतशीरपणे ओळख करणे समाविष्ट आहे. प्रोजेक्टच्या व्याप्ती, वेळापत्रक, खर्च किंवा गुणवत्तेवर परिणाम करू शकणाऱ्या सर्व संभाव्य जोखीम ओळखणे आणि त्यांचे दस्तऐवजीकरण करणे हे उद्दिष्ट आहे. या टप्प्यात वापरल्या जाणाऱ्या सामान्य तंत्रांमध्ये विचारमंथन सत्रे, तज्ञांच्या मुलाखती, डेल्फी तंत्र, SWOT विश्लेषण, दस्तऐवज पुनरावलोकने आणि ऐतिहासिक चेकलिस्टचा वापर यांचा समावेश आहे. संपूर्ण ओळख सुनिश्चित करण्यासाठी जोखीम सामान्यतः तांत्रिक, व्यवस्थापकीय, संघटनात्मक, आर्थिक आणि बाह्य अशा गटांमध्ये वर्गीकृत केल्या जातात. या प्रक्रियेचा परिणाम जोखीम नोंदणीमध्ये दस्तऐवजीकरण केला जातो, ज्यामध्ये ओळखल्या जाणाऱ्या जोखमींची यादी तसेच त्यांची कारणे, संभाव्य परिणाम आणि श्रेणी यासारख्या तपशीलांचा समावेश असतो. प्रभावी जोखीम ओळख प्रोजेक्ट संघांना सक्रिय धोरणे तयार करण्यास, अनिश्चितता कमी करण्यास आणि एकूण प्रोजेक्ट नियंत्रण वाढविण्यास मदत करते.

4.5.2 जोखीम विश्लेषण आणि व्यवस्थापन: जोखीम मूल्यांकन.

जोखीम मूल्यांकन ही जोखीम विश्लेषण आणि व्यवस्थापन प्रक्रियेतील दुसरी महत्त्वाची पायरी आहे. जोखीम ओळखल्यानंतर, जोखीम मूल्यांकनाचा उद्देश प्रत्येक जोखीमचे विश्लेषण आणि मूल्यांकन करणे आहे जेणेकरून त्याचा प्रोजेक्टवरील संभाव्य परिणाम समजून घेता येईल आणि त्याची शक्यता निश्चित करता येईल. हे जोखमींना प्राधान्य देण्यास मदत करते जेणेकरून प्रोजेक्ट व्यवस्थापक सर्वात गंभीर जोखमींवर लक्ष केंद्रित करू शकतील.

गुणात्मक आणि/किंवा परिमाणात्मक पद्धती वापरून जोखीम मूल्यांकन केले जाऊ शकते. गुणात्मक जोखीम मूल्यांकनात, संभाव्यता (कमी, मध्यम, उच्च) आणि प्रभाव (किरकोळ, मध्यम, गंभीर) सारख्या व्यक्तिनिष्ठ निकषांवर आधारित जोखीमांचे मूल्यांकन केले जाते. या दृष्टिकोनात संभाव्यता-प्रभाव मॅट्रिक्स आणि जोखीम रेटिंग स्केल सारखी साधने वापरली जातात. परिमाणात्मक जोखीम मूल्यांकनात, अपेक्षित मौद्रिक मूल्य (EMV), मोंटे कार्लो सिम्युलेशन किंवा संवेदनशीलता विश्लेषण सारख्या मॉडेल्सचा वापर करून जोखमींच्या संभाव्यता आणि आर्थिक किंवा वेळ-आधारित परिणामांचा अंदाज घेण्यासाठी संख्यात्मक पद्धती लागू केल्या जातात.

4.5.3 जोखीम विश्लेषण आणि व्यवस्थापन: जोखीम व्यवस्थापन आणि देखरेख.

जोखीम व्यवस्थापन आणि देखरेख हा जोखीम विश्लेषण आणि व्यवस्थापन प्रक्रियेतील अंतिम आणि चालू टप्पा आहे. जोखीम ओळखल्यानंतर आणि त्यांचे मूल्यांकन केल्यानंतर, जोखीमांना प्रतिसाद देण्यासाठी आणि प्रोजेक्टच्या संपूर्ण जीवनचक्रात त्यांचे सतत निरीक्षण करण्यासाठी धोरणे विकसित करण्यावर लक्ष केंद्रित केले जाते.

जोखीम व्यवस्थापनामध्ये प्रत्येक प्राधान्य दिलेल्या जोखमीसाठी योग्य प्रतिसाद निवडणे समाविष्ट आहे. नकारात्मक जोखीम (धोक्या) साठी सामान्य धोरणांमध्ये हे समाविष्ट आहे:

- टाळणे (जोखीम दूर करण्यासाठी योजना बदलणे),
- कमी करणे (संभाव्यता किंवा परिणाम कमी करणे),
- हस्तांतरण (जोखीम तृतीय पक्षाकडे हलवणे, उदा., विम्याद्वारे), आणि
- स्वीकृती (जोखीम स्वीकारणे आणि ती उद्भवल्यास त्याचा प्रभाव व्यवस्थापित करण्याची तयारी करणे).

सकारात्मक जोखीम (संधी) साठी, धोरणांमध्ये हे समाविष्ट असू शकते:

- शोषण (संधी साकार झाली आहे याची खात्री करणे)
- वाढ (संभाव्यता किंवा परिणाम वाढवणे)
- सामायिकरण (फायदा घेण्यासाठी इतरांसोबत भागीदारी करणे)
- स्वीकृती (जर ती सक्रिय नियोजनाशिवाय आली तर फायदा घेणे)

एकदा प्रतिसाद अंमलात आणले की, जोखीम निरीक्षण हे सुनिश्चित करते की जोखीमांचा मागोवा घेतला जातो, नवीन जोखीम ओळखल्या जातात आणि प्रतिसाद धोरणे प्रभावी राहतात. यामध्ये नियमित जोखीम पुनरावलोकने, जोखीम

नोंदणीमध्ये अद्यतने आणि स्थिती अहवाल समाविष्ट असतात. जर धोका घटना घडली तर आकस्मिक योजना किंवा फॉलबॅक धोरणे सक्रिय केली जातात.

4.5.4 जोखीम विश्लेषण आणि व्यवस्थापन: जोखीम शुद्धीकरण आणि कमी करणे.

जोखीम शुद्धीकरण आणि कमी करणे हा जोखीम विश्लेषण आणि व्यवस्थापन प्रक्रियेचा एक आवश्यक भाग आहे, जो ओळखल्या गेलेल्या जोखमींची समज सुधारण्यावर आणि त्यांचा प्रभाव किंवा शक्यता कमी करण्यासाठी सक्रिय पावले उचलण्यावर केंद्रित आहे. जोखीम शुद्धीकरणामध्ये व्यापक किंवा अस्पष्ट जोखीम अधिक विशिष्ट, सु-परिभाषित घटकांमध्ये विभागणे समाविष्ट आहे. ही प्रोसेस प्रत्येक जोखमीची मूळ कारणे आणि संभाव्य ट्रिगर्स ओळखण्यास मदत करते, ज्यामुळे त्यांचे प्रभावीपणे मूल्यांकन आणि व्यवस्थापन करणे सोपे होते. जोखीम शुद्धीकरण करून, प्रोजेक्ट संघांना धोका कसा, केव्हा आणि का येऊ शकतो याबद्दल स्पष्ट अंतर्दृष्टी मिळू शकते, ज्यामुळे निर्णय घेण्याची क्षमता आणि नियोजन वाढते. दुसरीकडे, जोखीम कमी करणे म्हणजे जोखीम होण्याची शक्यता कमी करणे किंवा तो झाल्यास त्याचा प्रभाव कमी करणे या उद्देशाने धोरणांचा विकास आणि अंमलबजावणी होय. सामान्य कमी करण्याच्या धोरणांमध्ये संवाद सुधारणे, चाचणी आणि गुणवत्ता हमी वाढवणे, रिडंडन्सी जोडणे, प्रशिक्षण आयोजित करणे किंवा वेळापत्रक आणि संसाधने समायोजित करणे समाविष्ट आहे. उदाहरणार्थ, जर एखाद्या गंभीर तंत्रज्ञानाच्या अपयशाचा धोका असेल तर कमी करण्यासाठी बॅकअप सिस्टम वापरणे किंवा लवकर प्रोटोटाइप चाचणी करणे समाविष्ट असू शकते. एकत्रितपणे, जोखीम शुद्धीकरण आणि कमी करणे हे सुनिश्चित करते की जोखीम केवळ ओळखल्या जातात आणि त्यांचे मूल्यांकन केले जात नाही तर ते अचूक आणि लक्ष्यित पद्धतीने सक्रियपणे व्यवस्थापित देखील केले जातात. यामुळे प्रोजेक्टची लवचिकता वाढते आणि अनिश्चितता प्रोजेक्टच्या उद्दिष्टांना अडथळा आणत नाही याची खात्री करण्यास मदत होते.

4.5.5 जोखीम विश्लेषण आणि व्यवस्थापन: RMMM योजना.

4.5.5.1 RMMM योजना म्हणजे जोखीम कमी करणे, देखरेख करणे आणि व्यवस्थापन योजना. हा एक संरचित दस्तऐवज आहे जो प्रोजेक्टच्या संपूर्ण जीवनचक्रात संभाव्य जोखमींना कसे सामोरे जाईल हे दर्शवितो. RMMM योजना ही प्रोजेक्टच्या एकूण जोखीम व्यवस्थापन धोरणाचा एक महत्त्वाचा घटक आहे.

4.5.5.2 RMMM योजनेचा उद्देश

RMMM योजना यासाठी डिझाइन केली आहे:

- जोखीम ओळखणे आणि प्राधान्य देणे
- जोखीम कमी करण्यासाठी (कमी) धोरणे विकसित करणे
- प्रोजेक्टदरम्यान जोखीमांचे निरीक्षण करणे
- जोखीम प्रत्यक्षात आल्यास प्रतिसाद व्यवस्थापित करणे

4.5.5.3 RMMM योजनेचे घटक

1. जोखीम ओळखणे
    - o प्रोजेक्टवर परिणाम करू शकणाऱ्या सर्व संभाव्य जोखमींची तपशीलवार यादी बनवणे.
    - o वर्णन, श्रेणी आणि मूळ कारणे समाविष्ट आहेत.
2. जोखीम मूल्यांकन
    - o संभाव्यता आणि परिणामाच्या दृष्टीने प्रत्येक जोखमीचे मूल्यांकनb करणे.
    - o जोखीम श्रेणीबद्ध किंवा वर्गीकृत केल्या जातात (उदा., उच्च, मध्यम, कमी).
3. जोखीम कमी करणे
    - o प्रत्येक जोखमीची शक्यता किंवा तीव्रता कमी करण्यासाठी नियोजित विशिष्ट कृती किंवा धोरणे आखणे.
    - o उदाहरण: मानवी चुकांचा धोका कमी करण्यासाठी कर्मचाऱ्यांना प्रशिक्षण देणे.
4. जोखीम देखरेख
    - o कालांतराने जोखीमांचा मागोवा घेण्याची, त्यांचे पुनर्मूल्यांकन करण्याची आणि नवीन जोखीम ओळखण्याची योजना करणे.
    - o नियोजित जोखीम पुनरावलोकने आणि जोखीम नोंदणीमध्ये अद्यतने समाविष्ट असू शकतात.

5. जोखीम व्यवस्थापन/आकस्मिक योजना
    - o जोखीम प्रत्यक्षात उद्भवल्यास अंमलात आणण्यासाठी फॉलबॅक किंवा प्रतिसाद योजना.
    - o व्यत्यय कमी करण्यास आणि प्रोजेक्ट ट्रॅकवर ठेवण्यास मदत करते.

4.5.5.4 आर एम एम एम योजनेचे फायदे (Advantages):

- प्रतिक्रियात्मक व्यवस्थापनाऐवजी सक्रिय व्यवस्थापनाला प्रोत्साहन देते.
- संघाची जागरूकता आणि तयारी सुधारते.
- चांगले निर्णय घेण्यास आणि संसाधनांचे वाटप सुनिश्चित करण्यास मदत करते.
- प्रोजेक्टच्या यशाची शक्यता वाढवते.

References:

1. Software Engineering: A practitioner's approach by Roger S. Pressman & Bruce R. Maxim, McGraw Hill Higher Education, New Delhi, (Ninth Edition) ISBN 93-5532-504-5
2. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
3. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
4. International standards (e.g., ISO 9001, IEEE 730, CMMI documentation)
5. Software Quality Assurance by Daniel Galin

Websites:

1. https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/
2. https://www.tutorialspoint.com/software_engineering/index.htm
3. IEEE 730 – Software Quality Assurance Standard - https://standards.ieee.org/
4. ISO 9001 for Software Quality - https://www.iso.org/standard/62085.html
5. ISACA / CMMI – https://www.isaca.org/enterprise/cmmi

युनिट-5
सॉफ्टवेअर प्रोजेक्ट मॅनेजमेंट
(Software Project Management)

विषय निष्पत्ती (Course Outcome):

CO5: सॉफ्टवेअर डेव्हलपमेंटमध्ये प्रोजेक्ट मॅनेजमेंट तंत्रे लागू करा.

घटक निष्पत्ती (Theory Learning Outcome - TLO):

1. सॉफ्टवेअर प्रोजेक्टसाठी सीपीएम/पीईआरटी शेड्यूलिंग तंत्र वापरा.
2. दिलेल्या सॉफ्टवेअर प्रोजेक्टच्या प्रगतीचा मागोवा घेण्यासाठी टाईमलाइन चार्ट किंवा गॅन्ट चार्ट तयार करा.

5.1 प्रोजेक्ट शेड्यूलिंग

प्रोजेक्ट शेड्यूलिंग हे अत्यंत महत्त्वाचे कार्य आहे. ठरवलेल्या वेळेत प्रोजेक्ट पूर्ण करणे कठीण असते. सॉफ्टवेअर उत्पादने विकसित करताना अनेक वेळा विलंब होतो. कधी मनुष्यबळाच्या अडचणी, कधी तांत्रिक समस्या, कधी डिझाइनच्या समस्या किंवा इतर काही कारणे यासाठी जबाबदार असतात. खाली अशा मूळ कारणांची यादी दिली आहे:

1. प्रोजेक्टमध्ये सहभागी नसलेल्या, आणि विकास प्रक्रियेतील अडथळ्यांचे ज्ञान नसलेल्या व्यक्ती अंतिम मुदती ठरवतात आणि त्या व्यक्ती व्यवस्थापक व विकासकांवर दबाव आणण्याचा प्रयत्न करतात.
2. ग्राहकाच्या मागणीत सतत होणारे बदल, जे वेळापत्रकात वेळोवेळी नोंदवले जात नाहीत.
3. काम पूर्ण करण्यासाठी लागणाऱ्या प्रयत्नांचे व संसाधनांचे अचूक मोजमाप न करणे.
4. प्रोजेक्ट सुरू करताना पूर्वानुमानित व अनपेक्षित जोखीम (रिस्कचा) विचारात न घेणे.
5. प्रोजेक्टचे शेड्युलिंग करण्यापूर्वी तांत्रिक अडचणी विचारात न घेणे.
6. अशाही काही मानव संसाधनासंबंधीत अडचणी असू शकतात ज्या वेळापत्रक तयार करताना विचारात घेतल्या गेल्या नाहीतवेळापत्रक तयार करताना मानव संसाधनांशी संबंधित अडचणींचा विचार न होणे.
7. प्रोजेक्ट कार्यसंघातील गैरसमज, जे विलंबाचे मोठे कारण ठरते ठरतात.
8. प्रोजेक्ट व्यवस्थापनाची अयशस्वी प्रतिक्रिया, म्हणजे प्रोजेक्ट वेळापत्रकात मागे पडत आहे हे लक्षात न येणे आणि त्यावर योग्य ती कृती न होणे.

सॉफ्टवेअर उद्योगात अनेक वेळा अंतिम मुदती अवास्तविक असतात, हे एक सत्य आहे. या मुदतींची मागणी बऱ्याचदा कायदेशीर कारणांमुळे केली जाते. प्रत्येक वेळी प्रोजेक्ट सुरू करताना, त्यासंबंधीच्या व्यावहारिक मर्यादांवर आधारित वेळापत्रक तयार करणे आवश्यक असते. अशा वेळापत्रकामुळे सॉफ्टवेअर कंपन्यांसाठी स्पर्धा करणे कठीण होते.
या सर्व समस्या टाळण्यासाठी, अशा परिस्थितीत पुढील स्टेप्सची शिफारस केली जाते:

1. इतिहासातील किंवा मागील डेटाचा उपयोग करून सविस्तर अंदाज तयार केला जाऊ शकतो. यामुळे प्रोजेक्टसाठी लागणारा अंदाजित वेळ व प्रयत्न शोधण्यात मदत होते.
2. सॉफ्टवेअर इंजिनीअरिंगसाठी रणनीती तयार करणे आवश्यक आहे, आणि ती वाढणारी प्रोसेस मॉडेल वापरून स्पष्ट परिभाषित केली पाहिजे. ही रणनीती क्रिटिकल कार्यक्षमता निश्चित मुदतीत वितरीत करण्यास सक्षम असावी आणि योजनेचे दस्तऐवजीकरण आवश्यक आहे.
3. सांगितलेल्या डेडलाईन मध्ये करणे का व्यवहारिक किंवा शक्य नाही यासाठी ग्राहकांसोबत बैठक घेणे आवश्यक असते
4. सध्याच्या डेडलाइननुसार प्रोजेक्ट पूर्ण करायचा असेल, तर किती टक्के सुधारणा आवश्यक आहे, हे ग्राहकाला स्पष्टपणे सांगणे आवश्यक आहे.
5. वैकल्पिक धोरण म्हणून वाढणारी विकासाची रणनीती सुचवावी.
6. प्रोजेक्ट वेळेत पूर्ण करण्यासाठी काही पर्याय उपलब्ध आहेत उदा., अतिरिक्त संसाधने मिळवण्यासाठी अर्थसंकल्पात वाढ करावी लागेल (जसे की लोक, यंत्रसामग्री इत्यादी). मात्र, प्रोजेक्ट गुंतागुंतीचा असेल आणि वेळ कमी असेल, तर त्याचा परिणाम खराब गुणवत्तेच्या उत्पादनात होऊ शकतो.

तांत्रिक प्रोजेक्टमध्ये शेकडो तांत्रिक टास्क्स असू शकतात. काही टास्क्स प्रोजेक्टमध्ये येतात, तर काही टास्क्स प्रोजेक्टबाहेर असू शकतात. काही विशिष्ट टास्क्स हे क्रिटिकल पाथवर येतात. जर हे टास्क्स शेड्यूलमध्ये विचारात घेतले नाहीत, तर प्रोजेक्ट कोसळू शकतो. प्रोजेक्ट मॅनेजरचे मुख्य काम म्हणजे प्रोजेक्टमध्ये येणाऱ्या सर्व टास्क्सची व्याख्या करणे, त्यांच्यातील परस्परावलंबन दर्शवणारे नेटवर्क तयार करणे आणि नेटवर्कमधील क्रिटिकल टास्क्स ओळखणे. त्यांनंतर, मॅनेजरने या टास्क्सच्या प्रगतीचा मागोवा घ्यावा लागतो, जेणेकरून 'प्रत्येक दिवसागणिक' होणारा विलंब ओळखता येईल. हे साध्य करण्यासाठी, प्रोजेक्ट मॅनेजरकडे असा अजेंडा किंवा शेड्यूल असणे आवश्यक आहे, जो इतक्या तपशीलात तयार केलेला असेल की त्याच्या आधारे प्रोजेक्टचे परीक्षण व नियंत्रण करता येईल. सॉफ्टवेअर प्रोजेक्ट शेड्यूलिंगमध्ये एक महत्त्वाची क्रिया म्हणजे - नियोजित कालावधीमध्ये एकूण अंदाजित प्रयत्नांचे वितरण, आणि ते विशिष्ट सॉफ्टवेअर इंजिनीअरिंग टास्क्समध्ये वाटप करून केले जाते. शेड्यूल वेळोवेळी विकसित होत असताना, हे दिसून येते की प्रोजेक्ट नियोजनाच्या सुरुवातीच्या टप्प्यात एक मॅक्रोस्कोपिक शेड्यूल तयार केले जाते. या मॅक्रोस्कोपिक शेड्यूलमध्ये सर्व प्रमुख प्रोसेस फ्रेमवर्क ॲक्टिव्हिटीज आणि मुख्य उत्पादन कार्ये समाविष्ट असतात, ज्यांच्यावर हे शेड्यूल लागू करता येते. शेड्यूल कसे तयार होते हे समजणे खूप सोपे आहे. एकदा मुख्य ॲक्टिव्हिटीज व फंक्शन्स शेड्यूल केल्या की, त्यानंतर प्रत्येक ॲक्टिव्हिटी व फंक्शन अधिक विस्ताराने लिहिले जाते आणि त्याचे तपशीलवार शेड्यूल तयार केले जाते. जसे आपण पाहिले आहे की प्रत्येक ॲक्टिव्हिटीमध्ये अनेक टास्क्स असतात, तसेच आपण प्रत्येक टास्कसाठी लागणारा वेळ निश्चित करू शकतो.

- सॉफ्टवेअर शेड्यूलिंगकडे दोन प्रकारांनी पाहिले जाऊ शकते:

प्रथम दृष्टीकोन:

संगणक सॉफ्टवेअरच्या रीलिझसाठी अंतिम तारीख आधीच जाहीर केलेली किंवा ठरवलेली असते.
(या परिस्थितीत, सॉफ्टवेअर संस्था निश्चित केलेल्या वेळेत प्रोजेक्ट पूर्ण करण्याच्या बंधनात असते.)

द्वितीय दृष्टीकोन:

सॉफ्टवेअर शेड्यूलिंग करताना क्रियांचा क्रम अंदाजे ठरवला जातो, पण सॉफ्टवेअरची रीलिझ तारीख सॉफ्टवेअर विकसित करणाऱ्या संस्थेद्वारे नंतर ठरवली जाते.
(या दृष्टीकोनात प्रयत्नांचे वितरण उपलब्ध संसाधनांचा योग्य उपयोग करण्यासाठी केले जाते, आणि सॉफ्टवेअरचे काळजीपूर्वक विश्लेषण केल्यानंतर अंतिम तारीख निश्चित केली जाते.)
व्यावहारिकदृष्ट्या पाहता, बहुतेक वेळा लोक दुसऱ्या दृष्टीकोनापेक्षा पहिल्या दृष्टीकोनाचेच अनुसरण करतात.

5.1.1 मूलभूत तत्त्वे (Basic Principles)

काही मूलभूत तत्त्वे सॉफ्टवेअर प्रोजेक्ट शेड्यूलिंगसाठी पाथदर्शक ठरतात:

1. **कंपार्टमेंटलायझेशन (Compartmentlization):** सॉफ्टवेअर प्रोजेक्टमध्ये अनेक व्यवस्थापित करता येतील अशा ॲक्टिव्हिटीज आणि टास्क्समध्ये विभागणी करणे आवश्यक आहे. कंपार्टमेंटलायझेशन साध्य करण्यासाठी प्रोडक्ट आणि प्रोसेस दोन्ही अधिक परिष्कृत केले जातात.
2. **इंटरडिपेंडनसी (Independancy):** प्रत्येक विभागलेली ॲक्टिव्हिटीकिंवा टास्क यांचे परस्परावलंबन निश्चित करणे आवश्यक असते. काही टास्क्स अनुक्रमे घडणे आवश्यक असतात, काही समांतर होऊ शकतात. काही ॲक्टिव्हिटीज त्या आधीच्या प्रोडक्टशिवाय सुरू होऊ शकत नाहीत, तर काही स्वतंत्रपणे पार पडू शकतात.
3. **टाईम अलोकेशन (Time Allocation):** प्रत्येक ॲक्टिव्हिटी लहान टास्क्समध्ये विभागली जाते. प्रत्येक टास्कसाठी वेळेचे वाटप करणे आवश्यक असते. कामाच्या वेळेचे वाटप युनिट्सच्या संख्येनुसार केले जाते. प्रत्येक टास्कला प्रारंभ तारीख व पूर्ण होण्याची तारीख दिली जाते. या तारखा त्यांच्या इंटरडिपेंडन्सीवर आधारित असतात आणि काम फुल-टाईम की पार्ट-टाईम असावे हे ठरवतात.
4. **एफर्ट व्हॅलिडेशन (Effort Validation):** सॉफ्टवेअर प्रोजेक्ट साठी लागणाऱ्या टीमची मनुष्य संख्या निश्चित उपलब्ध असते सॉफ्टवेअर प्रोजेक्टमध्ये सॉफ्टवेअर टीमवर निश्चित लोकसंख्या उपलब्ध असते. शेड्यूलिंग व टाईम अलोकेशन करताना मॅनेजरने याची खात्री करावी की उपलब्ध लोकांची संख्या नियोजित लोकसंख्येइतकीच असावी – त्यापेक्षा जास्त नसावी.

5. **डिफाइंड रिस्पॉन्सिबिलिटीज (Defined Responsibilities):** शेड्यूलमध्ये दिलेल्या प्रत्येक टास्कसाठी एक विशिष्ट टीम मेंबर जबाबदार असावा.
6. **डिफाइंड आऊटकम्स (Defined outcomes):** प्रत्येक शेड्यूल केलेल्या टास्कचा स्पष्ट आउटकम (output) असावा. सॉफ्टवेअर प्रोजेक्टसाठी हे आउटपुट बहुतेक वेळा एखादे वर्क प्रोडक्ट असते (उदा. एखाद्या घटकाचे डिझाइन).
7. **डिफाइंड माइलस्टोन्स (Defined Milestones):** प्रोजेक्ट माइलस्टोन म्हणजे एखाद्या ॲक्टिव्हिटी किंवा टास्क गटाने क्वालिटी कंट्रोल अंमलात आणताना गाठलेला ठराविक टप्पा. प्रत्येक टास्क किंवा टास्क गट एखाद्या माइलस्टोनशी जोडलेला असावा. माइलस्टोन तेव्हाच गाठलेला मानला जातो, जेव्हा एक किंवा अधिक वर्क प्रोडक्ट्सचे क्वालिटीसाठी रिव्ह्यू करून ते मंजूर केले गेलेले असतात.

5.1.2 वर्क ब्रेकडाउन स्ट्रक्चर (Work Breakdown Structure):

प्रोजेक्ट मॅनेजमेंटमधील WBS (वर्क ब्रेकडाउन स्ट्रक्चर) म्हणजे प्रोजेक्टचे श्रेणीबद्ध विघटन होय, ज्यामध्ये मोठ्या प्रोजेक्टला लहान घटकांमध्ये विभागले जाते. WBS ची सुरुवात प्रोजेक्टच्या एकूण उद्दिष्टांपासून होते, आणि त्यानंतर त्याचे विभाजन लहान, अधिक कार्यक्षम डिलिव्हरेबल्स मध्ये केले जाते. ही एक उपयुक्त डायग्रामॅटिक पद्धत आहे जी प्रोजेक्ट मॅनेजर्सना त्यांच्या प्रोजेक्टचा स्कोप स्पष्टपणे समजून घेण्यासाठी, तसेच पूर्ण प्रोजेक्टसाठी आवश्यक असलेली सर्व टास्क्स व्हिज्युअलाइझ करण्यास मदत करते. प्रोजेक्टमधील सर्व स्टेप्स आणि टास्क्स वर्क ब्रेकडाउन स्ट्रक्चर चार्टमध्ये स्पष्टपणे दर्शवले जातात, त्यामुळे ते एक अत्यावश्यक प्रोजेक्ट प्लॅनिंग टूल ठरते. WBS डायग्राममध्ये अंतिम डिलिव्हरेबल, तसेच त्याच्याशी संबंधित टास्क्स आणि वर्क पॅकेजेस वरच्या स्तरावर असतात. खालील WBS लेव्हल्स प्रोजेक्टच्या स्कोपचे उपविभाग करतात, ज्यामुळे प्रोजेक्ट सुरू होण्यापासून पूर्ण होईपर्यंत कोणते टास्क्स, डिलिव्हरेबल्स आणि वर्क पॅकेजेस आवश्यक आहेत हे स्पष्ट दिसून येते. प्रोजेक्ट मॅनेजर्स हे प्रोजेक्ट मॅनेजमेंट सॉफ्टवेअरचा वापर करून WBS तयार करतात आणि कार्यान्वित करतात. WBS स्तर आणि टास्क श्रेणीबद्धतेसह तयार केलेला गॅन्ट चार्ट वापरल्यास हे सॉफ्टवेअर प्रोजेक्टचे नियोजन, शेड्यूलिंग आणि अंमलबजावणीसाठी विशेषतः प्रभावी ठरते.

5.1.2.1 प्रोजेक्ट मॅनेजमेंटमधील WBS चा वापर (Use of WBS in Project Management):

प्रोजेक्ट शेड्यूल तयार करण्याची पहिली स्टेप म्हणजे WBS तयार करणे. WBS प्रोजेक्टची उद्दिष्टे आणि ध्येय साध्य करण्यासाठी पूर्ण करावयाची सर्व कामे कोणत्या क्रमाने करायची आहेत हे स्पष्टपणे परिभाषित करते. या पद्धतीने तुमचा प्रोजेक्ट व्हिज्युअलाइझ केल्यावर, तुम्ही प्रोजेक्टचा स्कोप समजू शकता आणि सर्व टास्क्ससाठी योग्य संसाधन वाटप करू शकता. एक नीट रचलेली वर्क ब्रेकडाउन स्ट्रक्चर (WBS) खालील महत्त्वाच्या प्रोजेक्ट मॅनेजमेंट प्रोसेस ग्रुप्स आणि नॉलेज एरियाजमध्ये मदत करते:

- प्रोजेक्ट प्लॅनिंग, प्रोजेक्ट शेड्यूलिंग आणि प्रोजेक्ट बजेटिंग
- रिस्क मॅनेजमेंट, रिसोर्स मॅनेजमेंट, टास्क मॅनेजमेंट आणि टीम मॅनेजमेंट

याव्यतिरिक्त, WBS प्रोजेक्ट मॅनेजमेंटमधील सामान्य अडचणी — जसे की: डेडलाइन चुकणे, स्कोप क्रिप (व्याप्ती अनियंत्रित वाढणे), आणि कॉस्ट ओव्हररन (खर्च अपेक्षेपेक्षा जास्त होणे) यांसारख्या समस्या टाळण्यास मदत करते.

वर्क ब्रेकडाउन स्ट्रक्चर (WBS) हे जटिल प्रोजेक्ट्समधून पाथदर्शन करणाऱ्या नकाशाप्रमाणे काम करते. तुमच्या प्रोजेक्ट स्कोपमध्ये अनेक फेजेस किंवा लहान सब-प्रोजेक्ट्स असू शकतात — आणि हे सब-प्रोजेक्ट्ससुद्धा टास्क्स, डिलिव्हरेबल्स आणि वर्क पॅकेजेसमध्ये विभागता येतात. WBS तुम्हाला या सर्व घटकांचे व्यवस्थापन करण्यात मदत करते आणि प्रोजेक्ट स्कोपमधील प्रत्येक बाब पूर्ण करण्यासाठी आवश्यक असलेल्या तपशीलांची स्पष्टता मिळवून देते.

वर्क ब्रेकडाउन स्ट्रक्चरचे उदाहरण:

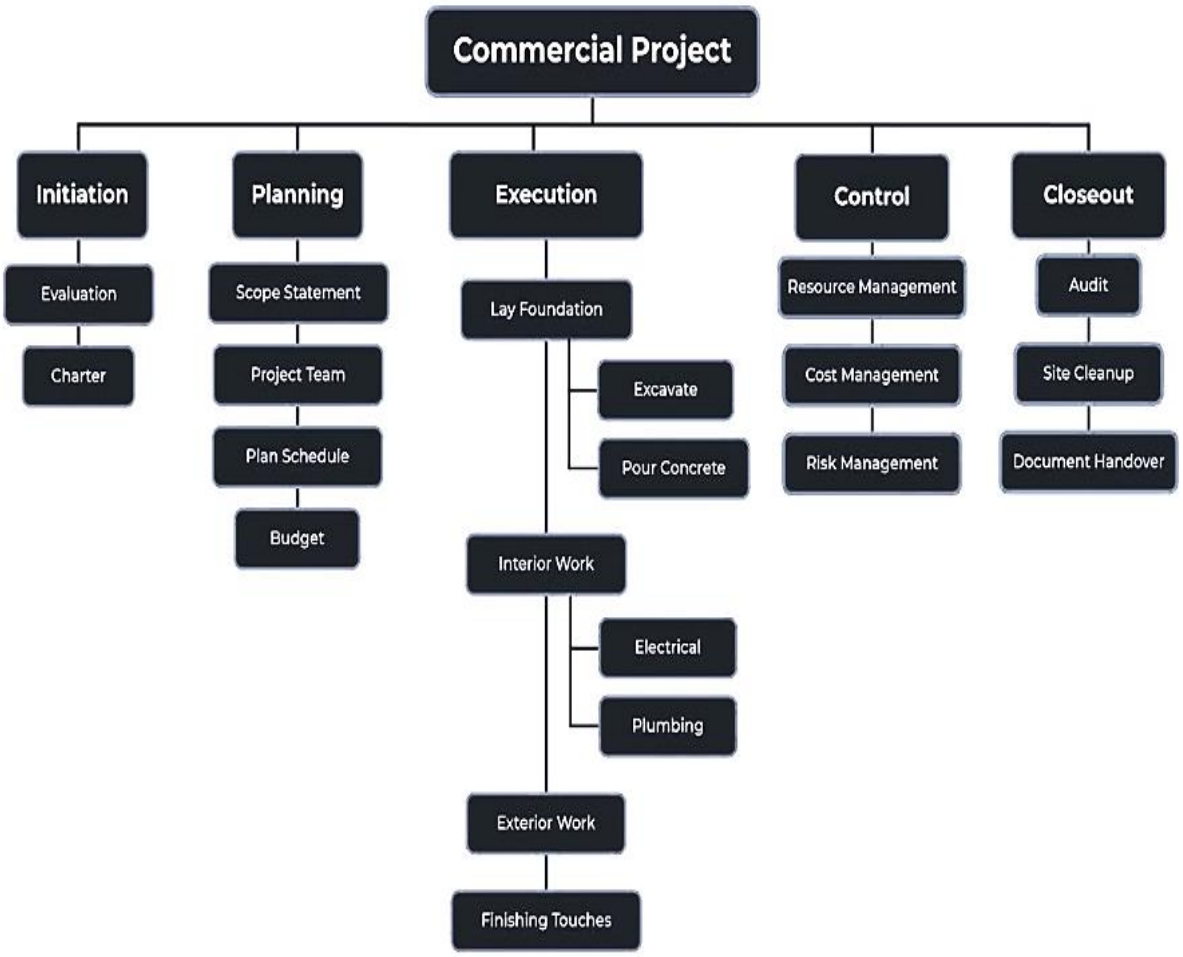


Fig. 5.1 वर्क ब्रेकडाउन स्ट्रक्चर (डब्ल्यूबीएस) (Work Breakdown Structure (WBS))

5.1.2.2 WBS चे प्रकार (Types of WBS):

WBS चे दोन मुख्य प्रकार असतात:

1. **डिलिव्हरेबल-आधारित WBS**
2. **फेज-आधारित WBS**

ते (WBS चे प्रकार) तुमचा प्रोजेक्ट वेळेच्या आधारावर विभागायचा आहे की स्कोपच्या (व्याप्तीच्या) आधारावर, यावर अवलंबून असतात.

5.1.2.2.1 डिलिव्हरेबल-आधारित वर्क ब्रेकडाउन स्ट्रक्चर (Deliverable-Based WBS):

डिलिव्हरेबल-आधारित WBS सर्वप्रथम प्रोजेक्टला त्याच्या स्कोपमधील प्रमुख भागांमध्ये कंट्रोल अकाउंट्स म्हणून विभागते, आणि नंतर त्यांचे पुढे डिलिव्हरेबल्स आणि वर्क पॅकेजेस मध्ये विघटन करते. कंट्रोल अकाउंट्स, वर्क पॅकेजेस आणि टास्क्स दर्शवणारे एक डिलिव्हरेबल-आधारित WBS उदाहरण खाली दिले आहे.

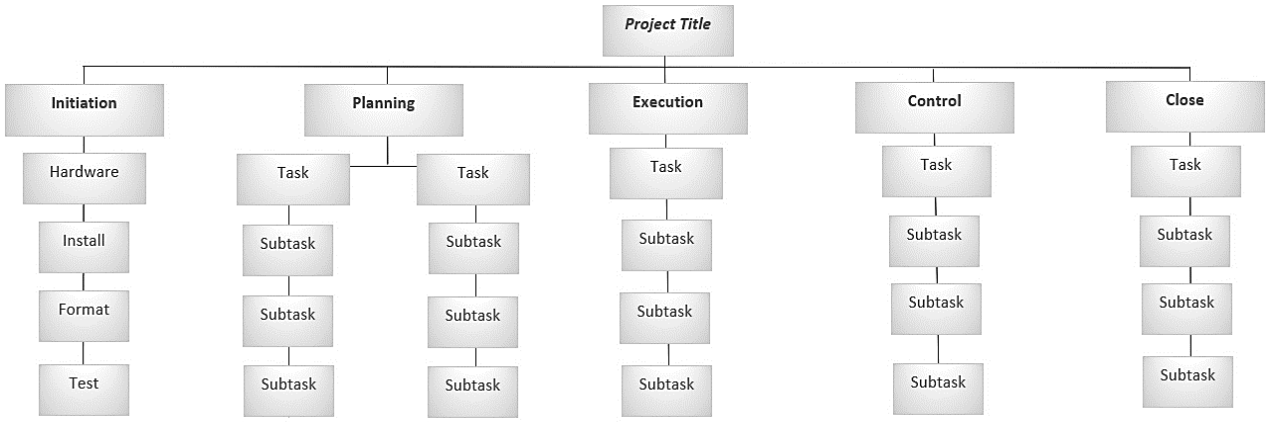


Fig. 5.2 डिलिव्हरेबल-बेस्ड डब्ल्यू बी एस (Deliverable-based WBS)

5.1.2.2.2 फेज-आधारित वर्क ब्रेकडाउन स्ट्रक्चर (Phase-Based Work Breakdown Structure):

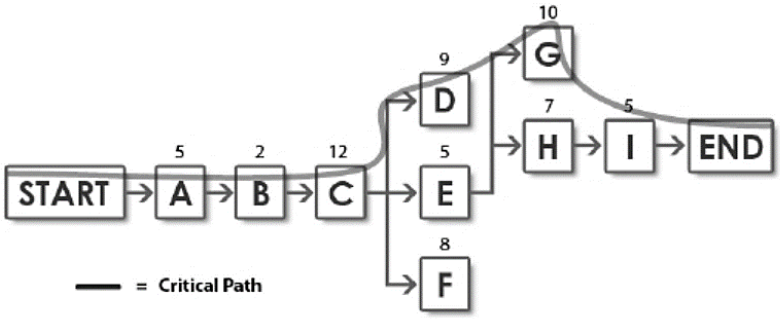
फेज-आधारित WBS मध्ये शीर्षस्थानी अंतिम डिलिव्हरेबल दर्शवले जाते, आणि त्याखालील WBS स्तरांमध्ये प्रोजेक्टचे पाच टप्पे दर्शवले जातात (सुरुवात, नियोजन, अंमलबजावणी, नियंत्रण आणि क्लोजआउट). डिलिव्हरेबल-आधारित WBS प्रमाणेच, या प्रोजेक्टच्या टप्प्यांचे पुढे प्रोजेक्ट डिलिव्हरेबल्स आणि वर्क पॅकेजेस मध्ये विघटन केले जाते. Fig. 5.1 मध्ये फेज-आधारित WBS चे उदाहरण दर्शवले आहे.

5.1.3 ॲक्टिव्हिटी नेटवर्क (Activity network) :

ॲक्टिव्हिटी नेटवर्क डायग्राम हे प्रोजेक्टमधील क्रियाकलापांचे एक डायग्राम आहे, जे नोड्स आणि ॲरोज वापरून क्रियाकलापांमधील अनुक्रमिक संबंध दर्शवते. ॲक्टिव्हिटी नेटवर्क डायग्राम टूलचा मोठ्या प्रमाणावर वापर केला जातो, आणि ते प्रोजेक्टच्या क्रिटिकल पाथ ओळखण्यासाठी अत्यावश्यक असते (जो प्रोजेक्टची अपेक्षित पूर्णता वेळ ठरवण्यासाठी वापरला जातो).

उदाहरणः समजा एका टीमला घर बांधण्याच्या प्रक्रियेत सुधारणा करण्याचे काम दिले जाते. ही टीम उत्खननाच्या टप्प्यापासून लँडस्केपींगपर्यंत असलेल्या सर्व महत्त्वाच्या स्टेप्सची यादी तयार करते.

A. Excavate	B. Foundation	C. Frame
D. Electrical	E. Roof	F. Masonry
G. Interior	H. Exterior	I. Landscape



टीम एक चार्ट तयार करते - ॲक्टिव्हिटी नेटवर्क डायग्राम - जिथे नोड्स (बॉक्सेस) हे घर बांधण्याच्या प्रक्रियेत येणाऱ्या नऊ प्रमुख स्टेप्सचे प्रतिनिधित्व करतात.

नोड्सना जोडणारे ॲरोज (बाण) प्रक्रियेचा प्रवाह दर्शवतात.

काही प्रोसेस स्टेप्स (नोड्स A, B आणि C) अनुक्रमे (सिरीजमध्ये) चालतात, तर इतर स्टेप्स (नोड्स D, E आणि F) समांतर (पॅरलल) चालतात.

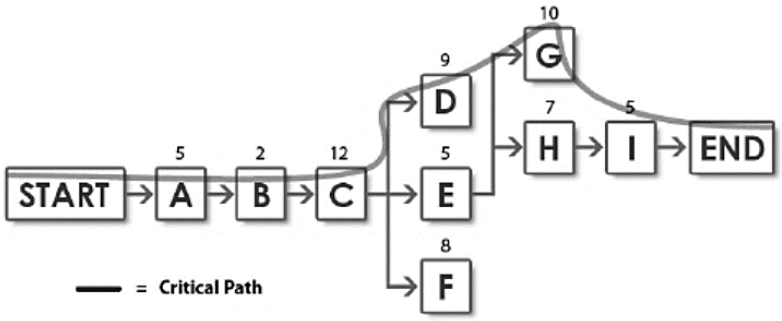
उदाहरणार्थ:

- स्टेप B तोपर्यंत सुरू होऊ शकत नाही, जोपर्यंत स्टेप A पूर्ण होत नाही.
- त्याचप्रमाणे, स्टेप C सुरू होण्यासाठी स्टेप B पूर्ण होणे आवश्यक आहे.

- तसेच, स्टेप D, E आणि F पूर्ण होईपर्यंत स्टेप H सुरू होऊ शकत नाही.
  म्हणजेच, स्टेप H सुरू होण्यासाठी D, E, F हे तिन्ही स्टेप्स पूर्ण झालेल्या असाव्यात.

म्हणून, नोड्स A, B आणि C हे सिरीजमध्ये चालतात, तर नोड्स D, E आणि F हे पॅरलल चालतात.
हे समजून घेणे महत्त्वाचे आहे कारण पॅरलल चालणाऱ्या स्टेप्सना पूर्ण होण्यासाठी लागणारा वेळ वेगवेगळा असू शकतो.

A. Excavate	5 days
B. Foundation	2 days
C. Frame	12 days
D. Electrical	9 days
E. Roof	5 days
F. Masonry	8 days
G. Interior	10 days
H. Exterior	7 days
I. Landscape	5 days



- **क्रिटिकल पाथ (Critical Path) :**

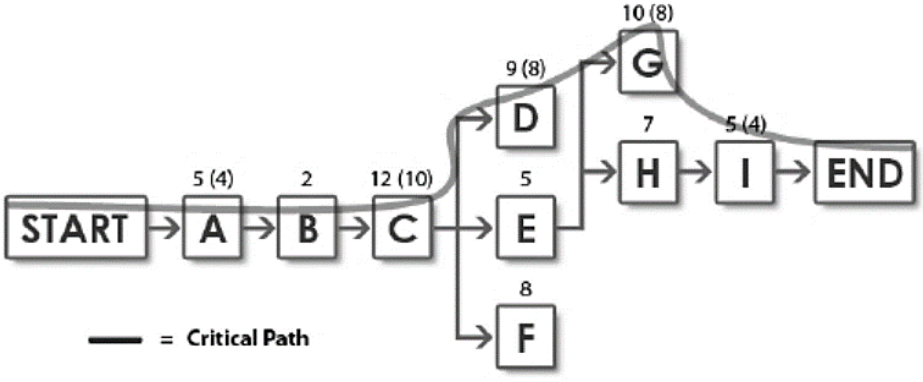
टीमचं काम म्हणजे D, E आणि F या नोड्सपैकी कोणता नोड सर्वाधिक वेळ घेईल आणि कोणता नोड कमी वेळ घेईल, हे ओळखणं. हे क्रिटिकल पाथ तयार करताना अत्यंत महत्त्वाचं असतं. उदाहरणार्थ, जर D नोड हे E आणि F यांच्या तुलनेत सर्वात जास्त वेळ घेणार असेल, तर D आणि E नोड्सना F सोबत एकाच वेळी सुरू होणं आवश्यक नाही. ते नंतर सुरू होऊ शकतात, पण ते सर्व स्टेप्स सर्वात जास्त वेळ घेणाऱ्या स्टेपच्या आधी पूर्ण झालेल्या असाव्यात. टीम नऊही स्टेप्सचं मूल्यांकन करते आणि प्रत्येक स्टेप किती दिवस घेईल यावर एकमत करते. क्रिटिकल पाथ ही अशी रेषा आहे जी सर्वात जास्त अपेक्षित वेळ लागणाऱ्या नोड्समधून जाते.

- **मोस्ट लाईकली टाईम (Most Likely Time):**

नोड्स A, B आणि C हे सिरीजमध्ये (एकामागोमाग एक) चालतात, त्यामुळे क्रिटिकल पाथ सरळ आणि स्पष्ट असतो.
पण लक्षात घ्या की D, E आणि F हे तीन नोड्स पॅरललमध्ये (एकाच वेळी) चालतात, आणि त्यामध्ये D नोड इतर दोन नोड्सच्या तुलनेत पूर्ण होण्यासाठी सर्वात जास्त वेळ घेणार आहे. म्हणूनच क्रिटिकल पाथ D आणि G या नोड्समधून जातो, कारण हे नोड्स सर्वात जास्त अपेक्षित वेळ घेणारे आहेत. वरील रेषा क्रिटिकल पाथ दाखवते. ॲक्टिव्हिटी नेटवर्क डायग्रामकडे पाहून टीम सहज लक्षात घेऊ शकते की क्रिटिकल पाथनुसार एकूण अपेक्षित पूर्ण होण्याचा कालावधी 50 दिवस आहे. (5 + 2 + 12 + 9 + 10 + 7 + 5 = 50 दिवस) हाच आहे मोस्ट लाईकली टाईम – म्हणजे काम पूर्ण होण्याची सर्वात शक्यता असलेली वेळ.

- **ऑप्टिमिस्टिक टाईम (Optimistic Time)**

वेळेच्या दृष्टीने सर्वोत्कृष्ट परिस्थिती (Best Case) काय असू शकते, हे टीमला जाणून घ्यायचं असू शकतं.
ही वेळ ठरवण्यासाठी, टीम प्रत्येक नोडसाठी शक्य तितका कमी वेळ निश्चित करते, आणि सर्व वेळा एकत्र करून एकूण वेळ काढते. कंसातील संख्या ही त्या-त्या नोडसाठीची सर्वात ऑप्टिमिस्टिक वेळ दर्शवते.
(4 + 2 + 10 + 8 + 8 + 7 + 4 = 43 दिवस)

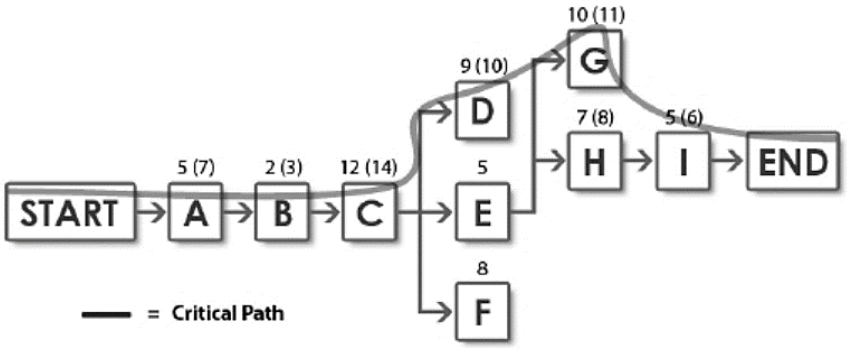


- **पिस्सीमिस्टिक टाईम (Pessimistic Time)**

वेळेच्या दृष्टीने सर्वात वाईट परिस्थिती (Worst Case) म्हणजेच पेस्सिमिस्टिक टाईम काय असू शकतो, हे जाणून घेण्याची गरज टीमला असू शकते.

ही वेळ ठरवण्यासाठी, टीम प्रत्येक नोडसाठी शक्य तितका जास्त वेळ ठरवते आणि सर्व वेळा एकत्र करून एकूण वेळ मोजते.

टीप: सर्वोत्कृष्ट (Optimistic) किंवा सर्वात वाईट (Pessimistic) वेळ ठरवताना क्रिटिकल पाथ फॉलो करणे आवश्यक असते. कंसातील संख्या या त्या-त्या नोड्ससाठीची सर्वात पेस्सिमिस्टिक वेळ दर्शवतात: (7 + 3 + 14 + 10 + 11 + 8 + 6 = 59 दिवस) लक्षात ठेवा: मोस्ट ऑप्टिमिस्टिक आणि पेस्सिमिस्टिक टाईम मोजताना फक्त क्रिटिकल पाथवर असलेल्या नोड्सचाच विचार केला जातो.



- **अपेक्षित वेळ (Expected Time)**

याचा अर्थ असा की प्रोजेक्ट पूर्ण होण्यासाठी सर्वात शक्यता असलेली वेळ 50 दिवस आहे, पण तो 59 दिवसांपर्यंत जाऊ शकतो, किंवा 43 दिवसांतही पूर्ण होऊ शकतो.

$$\text{Expected Time} = \frac{\text{Optimistic} + [4\ (\text{Most Likely})] + \text{Pessimistic}}{6} =$$

$$\text{Expected Time} = \frac{43+200+59}{6} = 50.3 \text{ days}$$

- **कंट्रोल बँड्स (Control Bands)**

आपण सरासरी वेळेच्या आसपास कंट्रोल बँड्सची गणना करू शकतो.

$$\text{Limits of expected variation} = \frac{\text{Optimistic - Pessimistic}}{6} =$$

$$\text{Limits of expected variation} = \frac{59-43}{6} =$$

$$\text{Limits of expected variation} = \frac{16}{6} = 2.7$$

क्रिटिकल पाथनुसार, प्रोजेक्ट पूर्ण होण्यासाठी लागणारी अपेक्षित वेळ सुमारे 47.6 दिवसांपासून 53.0 दिवसांपर्यंत असू शकते.

वरची मर्यादा: 50.3 + 2.7 = 53.0 दिवस

खालची मर्यादा: 50.3 – 2.7 = 47.6 दिवस

5.2 प्रोजेक्ट ट्रॅकिंग (Project Tracking):

प्रोजेक्ट शेड्यूलचा मागोवा घेण्यासाठी खालील उपाय वापरले जाऊ शकतात:

नियतकालिक प्रोजेक्ट स्टेटस मिटिंग घेणे, डेव्हलपमेंट लाईफ सायकलच्या सर्व टप्प्यांवर झालेल्या रिव्ह्यूजचे मूल्यांकन करणे, परिभाषित प्रोजेक्ट माइलस्टोन्स पूर्ण झाल्या आहेत का हे तपासणे, वास्तविक व नियोजित तारखांची तुलना करणे, आणि अर्न्ड व्हॅल्यू ॲनालिसिस (Earned Value Analysis) तंत्र वापरून प्रोग्रामचे परिमाणात्मक विश्लेषण करणे

सध्याच्या प्रोजेक्टच्या स्थितीचे मूल्यांकन करण्यासाठी एरर ट्रॅकिंग (Error Tracking) पद्धती देखील वापरता येतात. यासाठी मागील प्रोजेक्टमधील त्रुटींशी संबंधित मेट्रिक्स आणि मोजमाप गोळा करून, त्यांचा बेसलाइन म्हणून वापर केला जातो, जो रिअल टाईम डेटाशी तुलना करण्यासाठी वापरला जातो.

प्रोजेक्ट शेड्यूलिंग आणि ट्रॅकिंगची आवश्यकता:

1. त्रुटी ट्रॅकिंग पद्धती सध्याच्या प्रोजेक्टच्या स्थितीचे मूल्यांकन करण्यासाठी वापरता येतात. तसेच, इतिहासातील डेटाचा वापर करून प्रोजेक्टसाठी लागणाऱ्या प्रयत्नांचा व कालावधीचा तपशीलवार अंदाज लावता येतो.
2. अशा वाढीव प्रोसेस मॉडेलचा वापर करा, जे डेडलाइननुसार आवश्यक क्रिटिकल कार्यक्षमतेचे वितरण करेल, पण इतर विनंती केलेल्या कार्यक्षमतेस नंतर वितरित करण्याची मुभा ठेवेल.
3. ग्राहकांशी भेट घेऊन, तुमच्या पूर्वीच्या टीमच्या कामगिरीवर आधारित अंदाजाच्या मदतीने, डेडलाइन अवास्तव का आहे हे स्पष्टपणे समजावून सांगा.
4. संसाधने वाढवण्याचा पर्याय न निवडता, वाढीव विकास व वितरण धोरण पर्याय म्हणून सुचवा, किंवा वेळापत्रक डेडलाइनच्या पलीकडे सरकण्यास परवानगी द्या.

5.2.1 टाईमलाइन चार्ट (Timeline Charts) :

सॉफ्टवेअर प्रोजेक्टचे शेड्यूल तयार करताना, सुरुवात वर्क ब्रेकडाउन स्ट्रक्चरमधील टास्क सेटपासून केली जाते. स्वयंचलित टूल्सचा वापर केल्यास, हे वर्क ब्रेकडाउन टास्क नेटवर्क किंवा टास्क आऊटलाइन स्वरूपात इनपुट दिले जाते. त्यानंतर प्रत्येक टास्कसाठी प्रयत्न (Effort), कालावधी (Duration) आणि प्रारंभ तारीख (Start Date) दिली जाते. याशिवाय, विशिष्ट टीम सदस्यांना टास्क्स नियुक्त केले जाऊ शकतात. या इनपुटवरून, एक टाईमलाइन चार्ट (Time-line Chart) निर्माण होतो, ज्याला गॅन्ट चार्ट (Gantt Chart) असेही म्हणतात. टाईमलाइन चार्ट प्रत्येक प्रोजेक्ट फंक्शनसाठी किंवा प्रोजेक्टमध्ये काम करणाऱ्या प्रत्येक व्यक्तीसाठी वेगळा विकसित केला जाऊ शकतो.

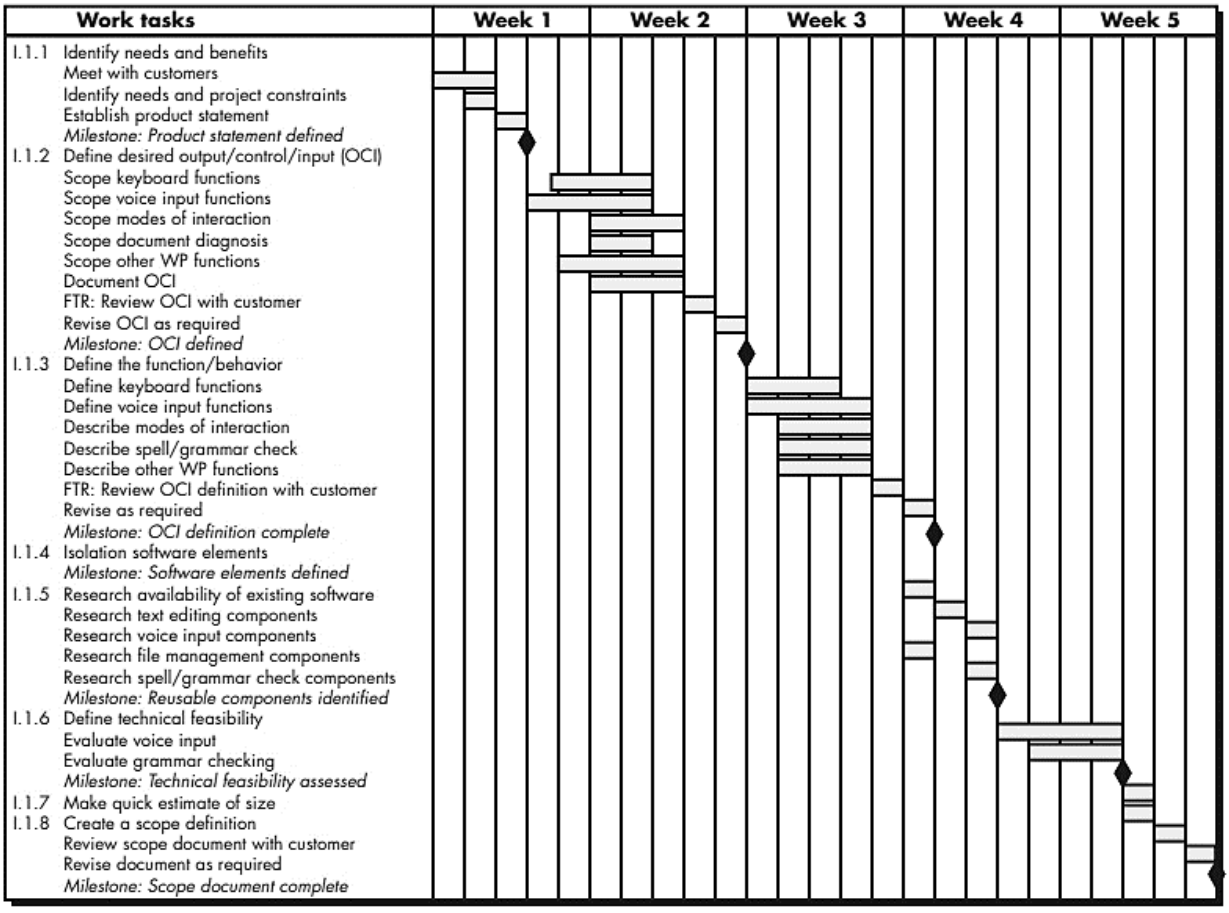


Fig.5.3: एक्झॅम्पल टाईम-लाइन चार्ट (An example time-line chart)

Fig. 5.3 मध्ये टाईमलाइन चार्टचे स्वरूप दर्शवले आहे. हे चार्ट वर्ड-प्रोसेसिंग (WP) सॉफ्टवेअर प्रोडक्टसाठी संकल्पना स्कोपिंग टास्कवर केंद्रित असलेल्या प्रोजेक्ट शेड्यूलचा एक भाग दर्शवते. संकल्पना स्कोपिंगशी संबंधित सर्व प्रोजेक्ट टास्क्स डाव्या बाजूच्या कॉलममध्ये सूचीबद्ध असतात. क्षैतिज पट्टे (bars) प्रत्येक टास्कचा कालावधी दर्शवतात. जेव्हा कॅलेंडरवर एकाच वेळी अनेक पट्टे दिसतात, तेव्हा टास्क कनकरन्सी (Concurrency) दर्शवली जाते. डायमंड्स (♦) हे माइलस्टोन्स दर्शवतात. एकदा टाईमलाइन चार्ट तयार करण्यासाठी आवश्यक माहिती इनपुट दिली की, बहुतेक प्रोजेक्ट शेड्यूलिंग टूल्स प्रोजेक्ट टेबल्स तयार करतात, ज्यामध्ये सर्व प्रोजेक्ट टास्क्स, त्यांची नियोजित व वास्तविक प्रारंभ व समाप्ती तारीख, तसेच इतर विविध संबंधित माहिती दिलेली असते (आकृती 5.4 मध्ये दर्शवले आहे). प्रोजेक्ट टेबल्ससह टाईमलाइन चार्टचा एकत्रित वापर केल्यास, प्रोजेक्ट मॅनेजरला शेड्यूलनुसार प्रगतीचा मागोवा घेणे व नियंत्रण ठेवणे शक्य होते.

Work tasks	Planned start	Actual start	Planned complete	Actual complete	Assigned person	Effort allocated	Notes
I.1.1 Identify needs and benefits							Scoping will
Meet with customers	wk1, d1	wk1, d1	wk1, d2	wk1, d2	BLS	2 p-d	require more
Identify needs and project constraints	wk1, d2	wk1, d2	wk1, d2	wk1, d2	JPP	1 p-d	effort/time
Establish product statement	wk1, d3	wk1, d3	wk1, d3	wk1, d3	BLS/JPP	1 p-d	
Milestone: Product statement defined	wk1, d3	wk1, d3	wk1, d3	wk1, d3			
I.1.2 Define desired output/control/input (OCI)							
Scope keyboard functions	wk1, d4	wk1, d4	wk2, d2		BLS	1.5 p-d	
Scope voice input functions	wk1, d3	wk1, d3	wk2, d2		JPP	2 p-d	
Scope modes of interaction	wk2, d1		wk2, d3		MLL	1 p-d	
Scope document diagnostics	wk2, d1		wk2, d2		BLS	1.5 p-d	
Scope other WP functions	wk1, d4	wk1, d4	wk2, d3		JPP	2 p-d	
Document OCI	wk2, d1		wk2, d3		MLL	3 p-d	
FTR: Review OCI with customer	wk2, d3		wk2, d3		all	3 p-d	
Revise OCI as required	wk2, d4		wk2, d4		all	3 p-d	
Milestone: OCI defined	wk2, d5		wk2, d5				
I.1.3 Define the function/behavior							

Fig. 5.4: एक्झॅम्पल प्रोजेक्ट टेबल (Example project table)

5.2.2 एअरनेड व्हॅल्यू ॲनालिसिस (Earned Value Analysis):

एअरनेड व्हॅल्यू ॲनालिसिस (EVA) हे एक प्रोजेक्ट परफॉर्मन्स मापन तंत्र आहे, जे स्कोप (व्याप्ती), वेळ आणि खर्च या तिन्ही डेटा घटकांचे एकत्रीकरण करते.कॉस्ट परफॉर्मन्स आणि प्रत्यक्ष साध्य केलेल्या कामाच्या तुलनेवर आधारित बेसलाइन वापरून, प्रोजेक्ट मॅनेजर "एअरनेड व्हॅल्यू" शोधू शकतो.बेसलाइन म्हणजे मूळ प्रोजेक्ट प्लॅन व त्याचा खर्च, जो WBS (वर्क ब्रेकडाउन स्ट्रक्चर) मध्ये मोजमाप युनिट्सच्या स्वरूपात दिलेला असतो.वास्तविक कामगिरीची तुलना या मूळ योजनेशी व संबंधित खर्चाशी केली जाते, आणि त्यातून मिळवलेले मूल्य (Earned Value) ठरवले जाते.

EVA मध्ये खालील तीन प्रमुख घटकांची गणना केली जाते:

- कार्य विभाजन संरचनेनुसार (WBS) नियोजित कामासाठीचाअंदाजित खर्च (BCWS).
- प्रत्यक्षात केलेल्या कामाचा वास्तविक खर्च (ACWP).
- प्रत्यक्षात पूर्ण केलेल्या कामासाठीचा अंदाजित खर्च (BCWP).

मिळवलेले मूल्य (EV) = BCWP = आजपर्यंतचा अंदाजित खर्च × पूर्ण कामाचा टक्केवारीत भाग

उदाहरण:

प्रोजेक्टच्या पहिल्या महिन्यात 10% काम पूर्ण होणे अपेक्षित होते, आणि त्यासाठीचा बजेटेड खर्च ₹10,00,000 होता. परंतु समजा प्रत्यक्षात केवळ 8% काम पूर्ण झाले.

म्हणूनच,

BCWS (Budgeted Cost of Work Scheduled) = ₹10,00,000

BCWP (Budgeted Cost of Work Performed / Earned Value) = ₹8,00,000

ACWP (Actual Cost of Work Performed) = ₹12,00,000

काही संस्था बिल करण्यायोग्य मूल्य (Billable Value) देखील मोजतात.

समजा ग्राहकासोबतच्या करारानुसार, बजेटच्या कामाशी संबंधित कालावधीसाठी बिल करण्यायोग्य रक्कम ₹10,00,000 होती.

पण प्रत्यक्षात केवळ 8% काम पूर्ण झाले, आणि त्यासाठीचे बिल करण्यायोग्य मूल्य ₹8,00,000 आहे.

बिल व्हेरिअन्स = ₹10,00,000 – ₹8,00,000 = ₹2,00,000

अपेक्षित उत्पन्नापेक्षा ₹2,00,000 कमी बिल करता येणार आहे.

आणि प्रत्यक्ष खर्च बजेटपेक्षा ₹4,00,000 अधिक झाला आहे.

किंवा

प्रगतीचे परिमाणात्मक विश्लेषण करण्यासाठी एक तंत्र उपलब्ध आहे, ज्याला "एअरनेड व्हॅल्यू ॲनालिसिस (EVA)" असे म्हणतात. एअरनेड व्हॅल्यू सिस्टीम प्रत्येक सॉफ्टवेअर प्रोजेक्ट टास्कसाठी एकसमान मूल्य स्केल प्रदान करते, कामाचा प्रकार काहीही असो. संपूर्ण प्रोजेक्टसाठी लागणाऱ्या एकूण तासांचा अंदाज लावला जातो, आणि प्रत्येक टास्कला त्याच्या टक्केवारीच्या आधारावर एक एअरनेड व्हॅल्यू दिली जाते. सोप्या शब्दांत सांगायचं झालं, तर एअरनेड व्हॅल्यू ही प्रगती मोजण्याची एक पद्धत आहे. हीपद्धत पूर्णतेची टक्केवारी "फीलिंगवर" अवलंबून न राहता, क्वांटिटेटिव्ह ॲनालिसिसद्वारे प्रोजेक्ट किती पूर्ण झाला आहे (percentage of completeness) हे अचूकपणे मोजू देते.

एअरनेड व्हॅल्यू ठरवण्यासाठी खालील पावले उचलली जातात:

1. शेड्यूलमध्ये दर्शवलेल्या प्रत्येक वर्क टास्कसाठी बजेटेड कॉस्ट ऑफ वर्क शेड्यूल (BCWS) निश्चित केला जातो. एस्टिमेशन दरम्यान, प्रत्येक सॉफ्टवेअर इंजिनीअरिंग टास्कसाठी लागणारे काम (पर्सन आवर्स किंवा पर्सन-डेजमध्ये) नियोजित केले जाते. म्हणूनच, BCWS म्हणजे त्या वर्क टास्कसाठी नियोजित एफर्ट होय.
   प्रोजेक्ट शेड्यूलमधील दिलेल्या एखाद्या विशिष्ट बिंदूवर प्रगती निश्चित करण्यासाठी, बीसीडब्ल्यूएसचे मूल्य म्हणजे त्या टप्प्यापर्यंत पूर्ण व्हायला हवे असलेल्या सर्व वर्क टास्कसाठीच्या BCWS मूल्यांची बेरीज.
2. सर्व वर्क टास्कसाठीची BCWS मूल्यं एकत्र करून बजेट ॲट कंप्लिशन (BAC) निश्चित केला जातो. म्हणूनच, सर्व वर्क टास्कसाठी: $BAC = \sum (BCWS_k)$
3. पुढे, बजेटेड कॉस्ट ऑफ वर्क परफॉर्म्ड (BCWP) चे मूल्य मोजले जाते. BCWP चे मूल्य म्हणजे प्रोजेक्ट शेड्यूलमधील एका विशिष्ट टप्प्यापर्यंत प्रत्यक्ष पूर्ण झालेल्या सर्व वर्क टास्कसाठीच्या BCWS मूल्यांची बेरीज.
   विल्केन्स असे नमूद करतो की: "BCWS आणि BCWP मधील मुख्य फरक असा आहे की, BCWS हे पूर्ण होण्यासाठी नियोजित असलेल्या ॲक्टिव्हिटीजच्या बजेटचं प्रतिनिधित्व करतं, तर BCWP हे प्रत्यक्षात पूर्ण झालेल्या ॲक्टिव्हिटीजच्या बजेटचं प्रतिनिधित्व करतं."

BCWS, BAC आणि BCWP ही मूल्यं वापरून पुढील महत्त्वाचे प्रगती निर्देशक (Progress Indicators) काढता येतात:

1. **शेड्यूल परफॉर्मन्स इंडेक्स (SPI)**
   SPI = BCWP / BCWS
2. **शेड्यूल व्हेरिअन्स (SV)**
   SV = BCWP – BCWS

SPI हे प्रोजेक्ट ने नियोजित संसाधनांचा किती कार्यक्षमतेने वापर केला आहे याचे निदर्शक आहे. SPI चे मूल्य जर 1.0 च्या जवळ असेल, तर प्रोजेक्ट शेड्यूल प्रभावीपणे अंमलात आणले जात आहे असे दर्शवते. SV म्हणजे नियोजित वेळापत्रकाच्या तुलनेत झालेला फरक, जो सोप्या शब्दांत एक निश्चित (absolute) मूल्य म्हणून दर्शवला जातो.

पूर्ण होण्यासाठी नियोजित टक्केवारी = BCWS / BAC

एखाद्या वेळ t पर्यंत किती टक्के काम पूर्ण व्हायला हवे होते, याचे निदर्शन करण्यासाठी वापरला जातो.
पूर्ण झालेली टक्केवारी = BCWP / BAC

एखाद्या विशिष्ट वेळ t ला प्रोजेक्ट किती टक्के पूर्ण झाला आहे, याचे परिमाणात्मक संकेत (quantitative indication) देतो.

काम केलेल्या वास्तविक किंमतीची (ACWP) गणना करणे देखील शक्य आहे. ACWP चे मूल्य म्हणजे प्रोजेक्ट शेड्यूलमध्ये एखाद्या विशिष्ट टप्प्यापर्यंत पूर्ण झालेल्या वर्क टास्क्सवर प्रत्यक्ष खर्च झालेल्या एफर्टची बेरीज. यानंतर पुढील गणना करणे शक्य होते.

किंमत कामगिरी निर्देशांक (Cost Performance Index - CPI):

CPI = BCWP / ACWP

खर्च भिन्नता (Cost Variance - CV):

CV = BCWP – ACWP

CPI चे मूल्य जर 1.0 च्या जवळ असेल, तर तो प्रोजेक्ट परिभाषित बजेटमध्ये आहे, याचा एक मजबूत संकेत मानला जातो.

CV हे प्रोजेक्टच्या विशिष्ट टप्प्यावर नियोजित खर्चाच्या तुलनेत खर्च बचतीचा किंवा जास्तीचा खर्च झाल्याचा परिपूर्ण (absolute) संकेत प्रदान करतं.

5.2.3 गॅन्ट चार्ट (Gantt Charts):

Generalized Activity Normalization Time Table (गॅन्ट) चार्ट हा चार्टचा एक प्रकार आहे, ज्यामध्ये क्षैतिज रेषांची मालिका असते जी त्या प्रोजेक्टसाठी नियोजित रकमेच्या तुलनेत दिलेल्या कालावधीत पूर्ण केलेल्या कामाचे किंवा उत्पादनाचे प्रमाण दर्शवते. हा क्षैतिज बार चार्ट हेन्री एल. गॅन्ट (अमेरिकन अभियंता आणि सामाजिक वैज्ञानिक) यांनी 1917 मध्ये उत्पादन नियंत्रण साधन म्हणून विकसित केला. याचा उपयोग शेड्यूलचे ग्राफिकल प्रतिनिधित्व करण्यासाठी केला जातो, जे प्रभावी नियोजन, कार्य समन्वय आणि प्रोजेक्टमधील विशिष्ट टास्क ट्रॅक करण्यास मदत करते. गॅन्ट चार्टचा मुख्य उद्देश म्हणजे प्रत्येक वैयक्तिक टास्कच्या व्याप्तीवर लक्ष केंद्रित करणे. म्हणून, टास्कचा संच गॅन्ट चार्टला इनपुट म्हणून दिला जातो. गॅन्ट चार्टला टाईमलाइन चार्ट असेही ओळखले जाते. हा संपूर्ण प्रोजेक्टसाठी विकसित केला जाऊ शकतो किंवा फक्त वैयक्तिक फंक्शन्ससाठी देखील तयार केला जाऊ शकतो. बहुतेक प्रोजेक्टमध्ये, टाईमलाइन चार्ट तयार केल्यानंतर प्रोजेक्ट टेबल्स तयार केल्या जातात. या टेबल्समध्ये सर्व टास्क्स त्यांची सुरूवातीची तारीख, समाप्ती तारीख आणि संबंधित माहितींसह योग्य क्रमाने सूचीबद्ध केलेले असतात.

गॅन्ट चार्ट खालील गोष्टींचे प्रतिनिधित्व करतो:

- सर्व टास्क्स डाव्या बाजूच्या स्तंभात (leftmost column) सूचीबद्ध केलेले असतात.
- आडव्या पट्ट्या (horizontal bars) संबंधित टास्क पूर्ण करण्यासाठी लागणारा टाईम दर्शवतात.
- जेव्हा कॅलेंडरवर एकाच वेळी अनेक आडव्या पट्ट्या दिसतात, तेव्हा त्याचा अर्थ असा की त्या टास्क्समध्ये कन्करन्सी (Concurrency) — म्हणजेच समांतरपणे काम — लागू करता येते.
- डायमंड्स (♦) हे माईलस्टोन्स दर्शवतात.

गॅन्ट चार्टचे फायदे (Advantages):

1. **प्रोजेक्ट सोपं करणे (Simplify Project):** गॅन्ट चार्टचा वापर मुख्यतः क्लिष्ट प्रोजेक्ट्स अधिक सोपं व समजण्यासारखं करण्यासाठी केला जातो.
2. **शेड्यूल स्थापन करणे (Establish Schedule):** गॅन्ट चार्ट प्रारंभिक प्रोजेक्ट शेड्यूल स्थापित करतो, ज्यामध्ये कोण काय करणार, केव्हा करणार आणि त्यासाठी किती वेळ लागेल हे स्पष्टपणे नमूद केलेले असते.
3. **कार्यक्षमता वाढवणे (Provide Efficiency):** यामुळे नियोजनात एफिशियन्सी वाढते आणि टीमला प्रोजेक्ट ॲक्टिव्हिटीजमध्ये उत्तम समन्वय साधता येतो.
4. **स्कोपवर भर देणे (Emphasize on Scope):** हे प्रत्येक वर्क टास्कच्या स्कोपवर लक्ष केंद्रित करण्यास मदत करतं, म्हणजे कार्याची मर्यादा आणि जबाबदाऱ्या स्पष्ट होतात.
5. **सोप्पा समजुतीसाठी (Ease of Understanding):** गॅन्ट चार्ट स्टेकहोल्डर्सना प्रोजेक्टची टाईमलाइन समजून घेणं सोपं करतं, आणि तारीखांमध्ये स्पष्टता आणतो.

6. **प्रोजेक्टचे दृश्यीकरण करणे (Visualize Project):** हे प्रोजेक्ट मॅनेजमेंट व संबंधित वर्क टास्क्सचे स्पष्ट दृश्यीकरण करण्यात मदत करतं.
7. **विचारांचे आयोजन आणि दृश्यमानता (Organize Thoughts and Highly Visible):** हे विचार सुटसुटीतपणे मांडते आणि प्रोजेक्ट सर्वांसाठी दृश्यमान ठेवते, जेणेकरून संस्थेतील प्रत्येक व्यक्तीला प्राथमिक समज मिळते, जरी ते प्रत्यक्ष प्रोजेक्टमध्ये सहभागी नसले तरी.
8. **व्यावहारिक आणि वास्तववादी नियोजन (Make Practical and Realistic Planning):** गॅन्ट चार्ट प्रोजेक्ट प्लॅनिंगला व्यावहारिक आणि वास्तववादी बनवतो, जेणेकरून उशीर आणि संभाव्य नुकसान टाळता येतं.

गॅन्ट चार्टचे तोटे (Disadvantages):

1. कधी कधी गॅन्ट चार्टचा वापर केल्याने प्रोजेक्ट अधिक क्लिष्ट होतो.
2. बार चार्टचा आकार हा नेहमीच प्रोजेक्टमध्ये पूर्ण झालेल्या कामाच्या प्रमाणाशी संबंधित नसतो.
3. गॅन्ट चार्ट आणि प्रोजेक्ट्स यांना नियमितपणे अपडेट करणं आवश्यक असतं.
4. गॅन्ट चार्ट एका कागदावर पाहणं शक्य नाही किंवा कठीण असतं. गॅन्ट चार्ट तयार करणाऱ्या सॉफ्टवेअर प्रॉडक्ट्स संगणक स्क्रीनवर पाहणं आवश्यक असतं, जेणेकरून संपूर्ण प्रोजेक्ट एकत्रितपणे पाहता येईल.

अनुप्रयोग (Applications):

काही असे व्यवसाय क्षेत्र आहेत जिथे गॅन्ट चार्टचा वापर अत्यंत फायदेशीर ठरतो. त्यापैकी काही खालीलप्रमाणे:

1. **ॲडव्हर्टायझिंग मॅनेजर (Advertising Manager):** ॲडव्हर्टायझिंग मॅनेजर्स सामान्यतः विज्ञापन कंपन्यांचे अंतिम उत्पादन नियंत्रित करतात व देखरेख करतात, विविध मीडियामध्ये जाहिरातींचं शेड्यूलिंग करतात इत्यादी.
2. **ऑपरेशन्स मॅनेजर (Operations Manager):** ऑपरेशन्स मॅनेजर्स सामान्यतः कंपनीच्या दैनंदिन कामकाजासाठी आवश्यक असलेल्या रिसोर्सेसचे नियंत्रण व व्यवस्थापन करतात.
3. **प्रोजेक्ट मॅनेजर (Project Manager):** प्रोजेक्ट मॅनेजर्स सहसा प्रोजेक्ट टीमचं मोटिवेशन करतात, टीम मेंबर्ससोबत समन्वय साधतात, टास्क्सचं शेड्यूलिंग करतात व ते वेळेत पूर्ण करतात, आणि स्टेकहोल्डर्सना रिपोर्टिंग करतात.

उदाहरण:

आजकाल अशा अनेक कंपन्या आणि कार्यसंघ आहेत जे त्यांच्या प्रोजेक्टचे नियोजन, शेड्यूलिंग आणि अंमलबजावणी करण्यासाठी गॅन्ट चार्ट वापरतात. त्यामध्ये मुख्यतः कन्सल्टिंग एजन्सीज, मॅन्युफॅक्चरिंग कंपन्या, मार्केटिंग टीम्स, कन्स्ट्रक्शन कंपन्या इत्यादींचा समावेश होतो. खाली गॅन्ट चार्टचे एक उदाहरण दिले आहे.

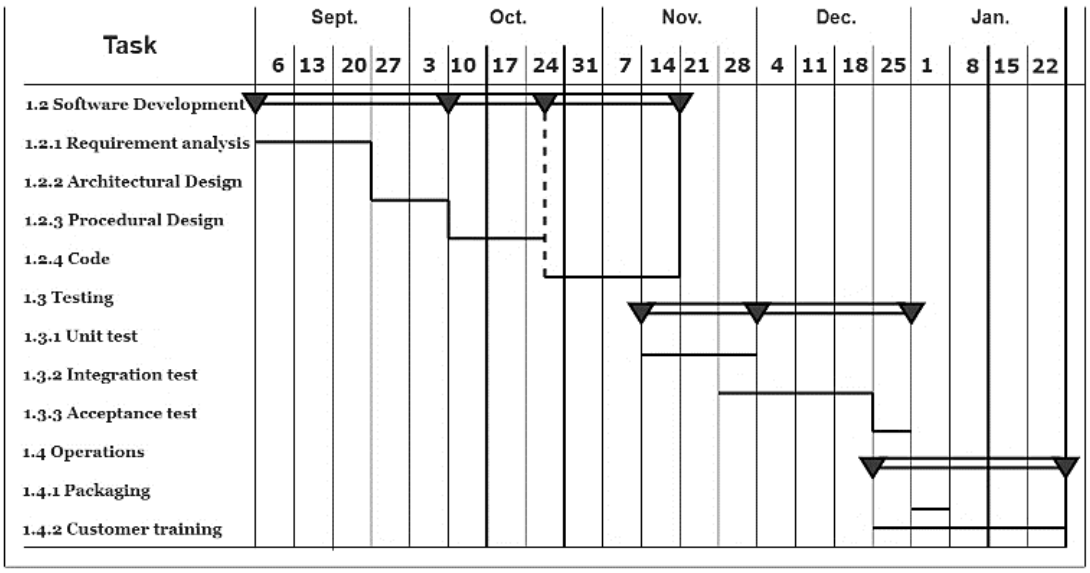


Fig. 5.5: गँट चार्ट (Gantt Chart)

5.3 शेड्यूलिंग तंत्र (Scheduling techniques):

5.3.1 क्रिटिकल पाथ मेथड (CPM)

क्रिटिकल पाथ मेथड (Critical Path Method - CPM) ही एक मेथड आहे जी प्रोजेक्ट नियोजनात वापरली जाते, मुख्यतः प्रोजेक्ट वेळेवर पूर्ण होण्यासाठी योग्य वेळापत्रक तयार करण्यासाठी. ही मेथड संपूर्ण प्रोजेक्ट कोणत्या किमान वेळेत पूर्ण होऊ शकतो हे ठरवण्यास मदत करते. या पद्धतीमध्ये दोन मुख्य संकल्पना आहेत – क्रिटिकल टास्क आणि क्रिटिकल पाथ. हे असे टास्क/ॲक्टिव्हिटी असते जे उशीरविना पूर्ण होणे आवश्यक असते, कारण त्यात उशीर झाल्यास संपूर्ण प्रोजेक्ट पूर्ण होण्यासही उशीर होतो. हे टास्क इतर अवलंबून असलेल्या टास्क सुरू होण्यापूर्वी वेळेत पूर्ण झाले पाहिजे. क्रिटिकल पाथ म्हणजे क्रिटिकल टास्क्स/ॲक्टिव्हिटीजचा अनुक्रम, जो प्रोजेक्ट नेटवर्कमधील सर्वात मोठा पाथ असतो. हा पाथ आपल्याला संपूर्ण प्रोजेक्ट पूर्ण करण्यासाठी लागणारा किमान वेळ दर्शवतो. क्रिटिकल पाथतील ॲक्टिव्हिटीजना क्रिटिकल ॲक्टिव्हिटीज म्हणतात, आणि जर या ॲक्टिव्हिटीजला उशीर झाला, तर संपूर्ण प्रोजेक्ट पूर्ण होण्यातही उशीर होतो.

प्रोजेक्ट मॅनेजमेंटमध्ये क्रिटिकल पाथ मेथड (CPM) वापरण्याचे फायदे (Advantages):

- प्रोजेक्ट्चे वेळापत्रक दृश्य स्वरूपात सादर करता येते.
- CPM चा वापर करून महत्वाच्या कार्यांवर लक्ष केंद्रित करता येते.
- CPM चा उपयोग रिस्क शोधण्यासाठी व हाताळण्यासाठी करता येतो.
- CPM प्रोजेक्ट कार्यसंघास उत्तम संवाद साधण्यास मदत करते.

प्रोजेक्टमध्ये क्रिटिकल पाथ कसा शोधायचा:

स्टेप 1: प्रोजेक्ट पूर्ण करण्यासाठी आवश्यक असलेली सर्व कार्ये ओळखा
स्टेप 2: कार्यांचा योग्य अनुक्रम ठरवा
स्टेप 3: प्रत्येक कार्याची कालावधी (duration) अंदाजित करा
स्टेप 4: नेटवर्क आकृती (Network Diagram) तयार करा
स्टेप 5: क्रिटिकल पाथ ओळखा
स्टेप 6: फ्लोट (Float) ची गणना करा
स्टेप 7: क्रिटिकल पथाचे सतत निरीक्षण करा

खालील तक्त्यामध्ये क्रियाकलापाचे लेबल, त्याची कालावधी (आठवड्यांमध्ये) आणि त्याचे पूर्वसर्ग (precedents) दिलेले आहेत. या माहितीचा वापर करून आपण क्रिटिकल पाथ आणि क्रिटिकल ॲक्टिव्हिटी शोधण्यासाठी क्रिटिकल पाथ मेथड (CPM) वापरणार आहोत.

Activity	Duration (in weeks)	Precedents
A	6	-
B	4	-
C	3	A
D	4	B
E	3	B
F	10	-
G	3	E, F
H	2	C, D

ॲक्टिव्हिटी-ऑन-नोड नेटवर्क आकृती तयार करण्यासाठी नियम:

- प्रोजेक्ट नेटवर्कमध्ये फक्त एक प्रारंभ नोड (Start Node) असावा
- प्रोजेक्ट नेटवर्कमध्ये फक्त एक समाप्ती नोड (End Node) असावा
- प्रत्येक नोडमध्ये काही कालावधी (Duration) असतो

- लिंक्स (Links) सामान्यतः कोणताही कालावधी नसतो
- "पूर्वसर्ग" (Precedents) म्हणजे लगेच आधी पूर्ण होणाऱ्या क्रियाकलापांचा उल्लेख
- प्रोजेक्ट नेटवर्कमध्ये वेळ डावीकडून उजवीकडे सरकते
- नेटवर्कमध्ये लूप (पुनरावृत्तीचे वळण) नसावेत
- नेटवर्कमध्ये डँगल्स (काटलेले किंवा अधुरे राहिलेले टोक) नसावेत

नोड रेप्रेसेंटेशन (Node Representation):

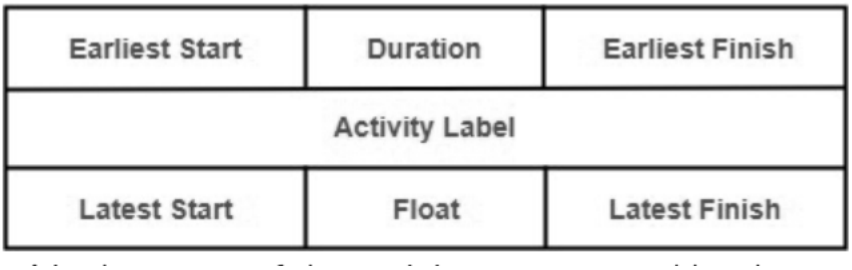


Fig. 5.6: नोड रिप्रेझेंटेशन (Node Representation)

- ॲक्टिव्हिटी लेबल म्हणजे त्या नोडद्वारे दर्शविलेल्या क्रियाकलापाचे नाव आहे.
- सर्वात लवकर प्रारंभ (Earliest Start) म्हणजे ती तारीख किंवा वेळ आहे ज्यावर ॲक्टिव्हिटी लवकरात लवकर सुरू केला जाऊ शकतो.
- सर्वात लवकर समाप्त (Earliest Finish) म्हणजे ती तारीख किंवा वेळ आहे ज्यावर ॲक्टिव्हिटी लवकरात लवकर पूर्ण केला जाऊ शकतो.
- सर्वात उशिरा प्रारंभ (Latest Start) म्हणजे ती तारीख किंवा वेळ आहे ज्यावर ॲक्टिव्हिटी उशिरात उशिरा सुरू केला जाऊ शकतो, तरीही प्रोजेक्ट वेळेत पूर्ण होईल.
- सर्वात उशिरा समाप्त (Latest Finish) म्हणजे ती तारीख किंवा वेळ आहे ज्यावर ॲक्टिव्हिटी उशिरात उशिरा पूर्ण केला जाऊ शकतो, तरीही प्रोजेक्ट वेळेत पूर्ण होईल.
- फ्लोट (Float) हा लवकर प्रारंभ आणि उशिरा प्रारंभ किंवा लवकर समाप्त आणि उशिरा समाप्त यामधील फरक असतो.

ॲक्टिव्हिटी-ऑन-नोड आकृती:

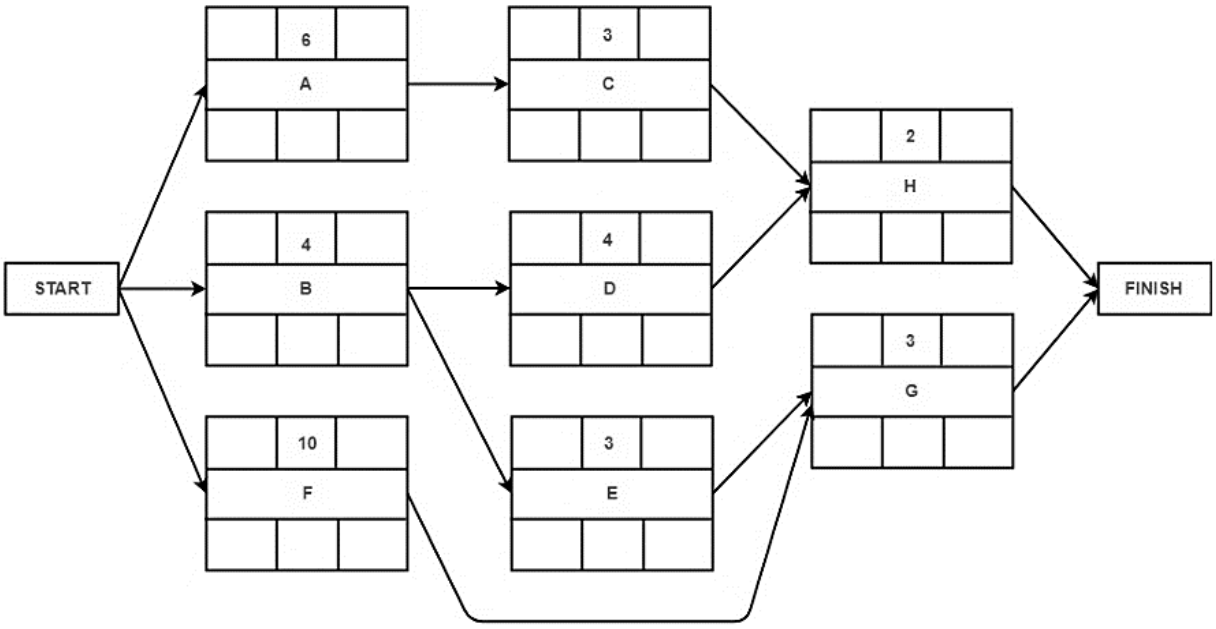


Fig. 5.7 ॲक्टिव्हिटी -ऑन-नोड डायग्राम (Activity-On-Node diagram)

5.3.1.1 प्रोजेक्ट मॅनेजमेंटमधील क्रिटिकल पाथमध्ये फॉरवर्ड पास (Forward Pass):

प्रत्येक ॲक्टिव्हिटी कोणत्या लवकरात लवकर तारखेला सुरू आणि पूर्ण होऊ शकतो याची गणना करण्यासाठी फॉरवर्ड पास वापरला जातो.

1. ॲक्टिव्हिटी A त्वरित सुरू होऊ शकतो. त्यामुळे त्याची सुरुवातीची सर्वात लवकर तारीख शून्य आहे, म्हणजेच ES(A) = 0. त्याची अंमलबजावणी पूर्ण होण्यासाठी 6 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 6 म्हणजेच EF(A) = 6.
2. ॲक्टिव्हिटी B त्वरित सुरू होऊ शकतो. त्यामुळे त्याची सुरुवातीची सर्वात लवकर तारीख शून्य आहे, म्हणजेच ES(B) = 0. त्याची अंमलबजावणी पूर्ण होण्यासाठी 4 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 4 म्हणजेच EF(B) = 4.
3. ॲक्टिव्हिटी F त्वरित सुरू होऊ शकतो. त्यामुळे त्याची सुरुवातीची सर्वात लवकर तारीख शून्य आहे, म्हणजेच ES(F) = 0. त्याची अंमलबजावणी पूर्ण होण्यासाठी 10 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 10 म्हणजेच EF(F) = 10.
4. ॲक्टिव्हिटी C, ॲक्टिव्हिटी A पूर्ण होताच सुरू होतो. त्यामुळे तो अंमलबजावणी सुरू करू शकतो अशी सर्वात लवकर तारीख आठवडा 6 आहे, म्हणजेच ES(C) = 6. त्याची अंमलबजावणी पूर्ण होण्यासाठी 3 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 9 म्हणजेच EF(C) = 9.
5. ॲक्टिव्हिटी D, ॲक्टिव्हिटी B पूर्ण होताच सुरू होतो. त्यामुळे तो अंमलबजावणी सुरू करू शकतो अशी सर्वात लवकर तारीख आठवडा 4 आहे, म्हणजेच ES(D) = 4. त्याची अंमलबजावणी पूर्ण होण्यासाठी 4 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 8 म्हणजेच EF(D) = 8.
6. ॲक्टिव्हिटी E, ॲक्टिव्हिटी B पूर्ण होताच सुरू होतो. त्यामुळे तो अंमलबजावणी सुरू करू शकतो अशी सर्वात लवकर तारीख आठवडा 4 आहे, म्हणजेच ES(E) = 4. त्याची अंमलबजावणी पूर्ण होण्यासाठी 3 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 7 म्हणजेच EF(E) = 7.
7. ॲक्टिव्हिटी G, ॲक्टिव्हिटी E आणि ॲक्टिव्हिटी F पूर्ण झाल्यानंतरच सुरू होतो. कारण या क्रियाकलापासाठी दोन्हींची अंमलबजावणी पूर्ण होणे आवश्यक आहे, त्यामुळे आपण MAX(ES(E), ES(F)) चा विचार करू. त्यामुळे तो अंमलबजावणी सुरू करू शकतो अशी सर्वात लवकर तारीख आठवडा 10 आहे, म्हणजेच ES(G) = 10. त्याची अंमलबजावणी पूर्ण होण्यासाठी 3 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 13 म्हणजेच EF(G) = 13.
8. ॲक्टिव्हिटी H, ॲक्टिव्हिटी C आणि ॲक्टिव्हिटी D पूर्ण झाल्यानंतरच सुरू होतो. कारण या क्रियाकलापासाठी दोन्हींची अंमलबजावणी पूर्ण होणे आवश्यक आहे, त्यामुळे आपण MAX(ES(C), ES(D)) चा विचार करू. त्यामुळे तो अंमलबजावणी सुरू करू शकतो अशी सर्वात लवकर तारीख आठवडा 9 आहे, म्हणजेच ES(H) = 9. त्याची अंमलबजावणी पूर्ण होण्यासाठी 2 आठवडे लागतात. त्यामुळे लवकरात लवकर तो पूर्ण होऊ शकतो आठवडा 11 म्हणजेच EF(H) = 11.

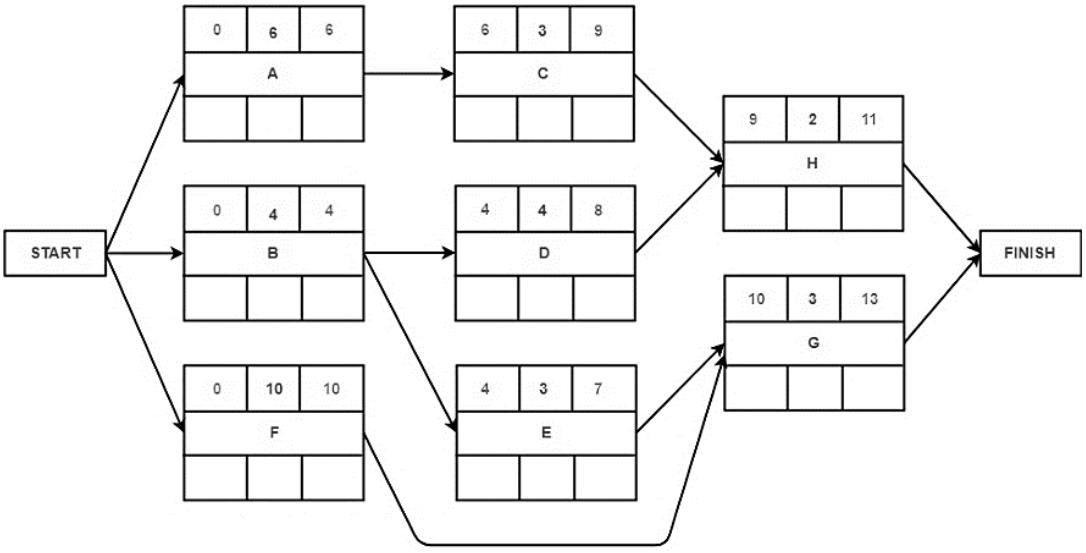


Fig. 5.8: फॉरवर्ड पास इन क्रिटिकल पाथ (Forward Pass in Critical path)

5.3.1.2 प्रोजेक्ट मॅनेजमेंटमधील क्रिटिकल पाथमध्ये बॅकवर्ड पास (Backward Pass):

प्रोजेक्टच्या शेवटच्या तारखेला विलंब न होता प्रत्येक ॲक्टिव्हिटी सुरू व पूर्ण केला जाऊ शकतो अशा नवीनतम तारखा (Latest Dates) शोधण्यासाठी बॅकवर्ड पास वापरला जातो. गृहितक: नवीनतम अंतिम तारीख = लवकरात लवकर अंतिम तारीख (प्रोजेक्टची).

- ॲक्टिव्हिटी G ची नवीनतम समाप्ती तारीख ही गृहित धरण्यानुसार प्रोजेक्टच्या समाप्तीपूर्वीच्या क्रियाकलापाची लवकरात लवकर समाप्ती तारीख आहे, म्हणजेच LF(G) = 13. त्याच्या अंमलबजावणीस 3 आठवडे लागतात. म्हणून, हा ॲक्टिव्हिटी सुरू होण्याची नवीनतम शक्य तारीख आठवडा 10 आहे, म्हणजेच LS(G) = 10.
- ॲक्टिव्हिटी H ची नवीनतम समाप्ती तारीख ही देखील गृहित धरण्यानुसार प्रोजेक्टच्या समाप्तीपूर्वीच्या क्रियाकलापाची लवकरात लवकर समाप्ती तारीख आहे, म्हणजेच LF(H) = 13. त्याच्या अंमलबजावणीस 2 आठवडे लागतात. म्हणून, हा ॲक्टिव्हिटी सुरू होण्याची नवीनतम शक्य तारीख आठवडा 11 आहे, म्हणजेच LS(H) = 11.
- ॲक्टिव्हिटी C साठी नवीनतम समाप्ती तारीख H या क्रियाकलापाची नवीनतम प्रारंभ तारीख असेल, म्हणजेच LF(C) = 11. त्याच्या अंमलबजावणीस 3 आठवडे लागतात. म्हणून, C हा ॲक्टिव्हिटी सुरू होण्याची नवीनतम शक्य तारीख आठवडा 8 आहे, म्हणजेच LS(C) = 8.
- ॲक्टिव्हिटी D साठी नवीनतम समाप्ती तारीख म्हणजे ॲक्टिव्हिटी H ची नवीनतम प्रारंभ तारीख असेल, म्हणजेच LF(D) = 11. या क्रियाकलापाच्या अंमलबजावणीस 4 आठवडे लागतात. त्यामुळे, तो सुरू होण्याची नवीनतम शक्य तारीख आठवडा 7 आहे, म्हणजेच LS(D) = 7.
- ॲक्टिव्हिटी E साठी नवीनतम समाप्ती तारीख म्हणजे ॲक्टिव्हिटी G ची नवीनतम प्रारंभ तारीख असेल, म्हणजेच LF(E) = 10. या क्रियाकलापाच्या अंमलबजावणीस 3 आठवडे लागतात. त्यामुळे, तो सुरू होण्याची नवीनतम शक्य तारीख आठवडा 7 आहे, म्हणजेच LS(E) = 7.
- ॲक्टिव्हिटी F साठी नवीनतम समाप्ती तारीख म्हणजे ॲक्टिव्हिटी G ची नवीनतम प्रारंभ तारीख असेल, म्हणजेच LF(F) = 10. या क्रियाकलापाच्या अंमलबजावणीस 10 आठवडे लागतात. त्यामुळे, तो सुरू होण्याची नवीनतम शक्य तारीख आठवडा 0 आहे, म्हणजेच LS(F) = 0.
- ॲक्टिव्हिटी A साठी नवीनतम समाप्ती तारीख म्हणजे ॲक्टिव्हिटी C ची नवीनतम प्रारंभ तारीख असेल, म्हणजेच LF(A) = 8. या क्रियाकलापाची अंमलबजावणी पूर्ण होण्यासाठी 6 आठवडे लागतात. त्यामुळे, तो सुरू होण्याची नवीनतम शक्य तारीख आठवडा 2 आहे, म्हणजेच LS(A) = 2.

- ॲक्टिव्हिटी B साठी नवीनतम समाप्ती तारीख म्हणजे ॲक्टिव्हिटी D आणि E याच्या नवीनतम प्रारंभ तारखांपैकी लवकरची तारीख, म्हणजेच LF(B) = 7. या क्रियाकलापाची अंमलबजावणी पूर्ण होण्यासाठी 4 आठवडे लागतात. त्यामुळे, तो सुरू होण्याची नवीनतम शक्य तारीख आठवडा 3 आहे, म्हणजेच LS(B) = 3.

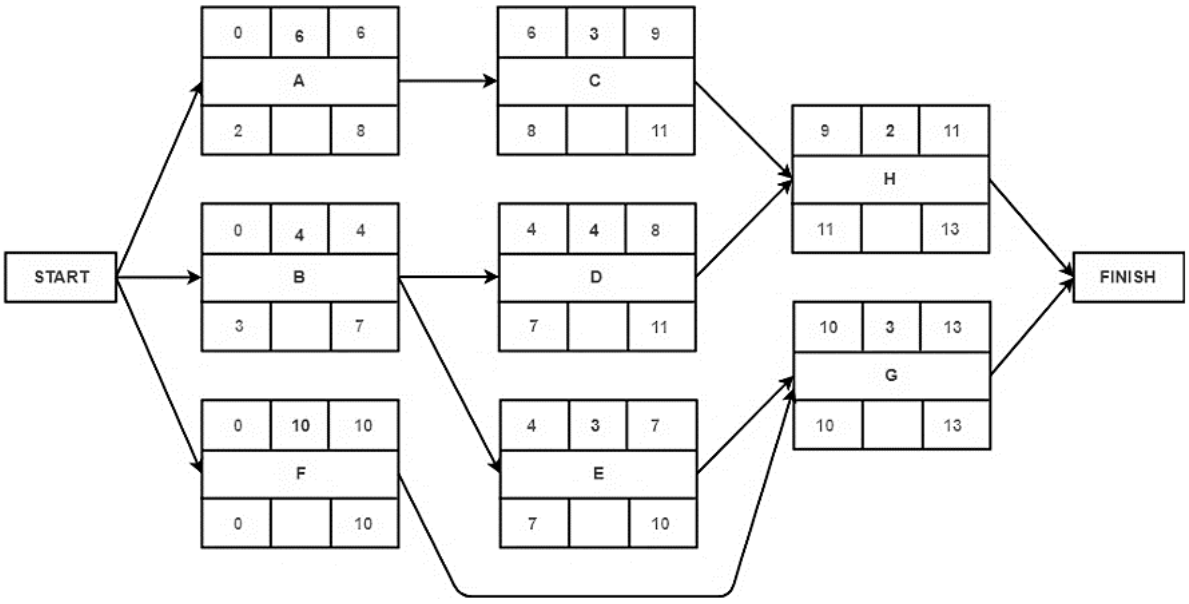


Fig. 5.9: बॅकवर्ड पास इन क्रिटिकल पाथ (Backward Pass in Critical path)

5.3.1.3 क्रिटिकल पाथ ओळखणे (Identifying Critical Path):

क्रिटिकल पाथ हा असा मार्ग आहे जो आपल्याला संपूर्ण प्रोजेक्ट लवकरात लवकर किती वेळेत पूर्ण होऊ शकतो याचा अंदाज लावण्यास मदत करतो. या क्रिटिकल पाथवरील कोणत्याही क्रियाकलापात विलंब झाल्यास संपूर्ण प्रोजेक्ट पूर्ण होण्यात विलंब होतो. क्रिटिकल पाथ ओळखण्यासाठी, प्रत्येक क्रियाकलापाचा फ्लोट (float) म्हणजेच उपलब्ध विलंबकालावधी शोधणे आवश्यक असते. फ्लोट म्हणजे एखाद्या क्रियाकलापाच्या: लवकरच्या प्रारंभ तारीख व नवीनतम प्रारंभ तारीखमधील फरक किंवा लवकरच्या समाप्ती तारीख व नवीनतम समाप्ती तारीखमधील फरक हा फ्लोट आपल्याला दर्शवतो की एखादा ॲक्टिव्हिटी प्रोजेक्टची अंतिम मुदत न चुकवता किती वेळ विलंब होऊ शकतो. जर एखाद्या क्रियाकलापाचा फ्लोट शून्य (0) असेल, तर तो ॲक्टिव्हिटी क्रिटिकल मानला जातो आणि तो क्रिटिकल पाथमध्ये समाविष्ट केला जातो. या उदाहरणात, F आणि G हे ॲक्टिव्हिटी शून्य फ्लोट असलेले असल्यामुळे ते क्रिटिकल ॲक्टिव्हिटी आहेत.

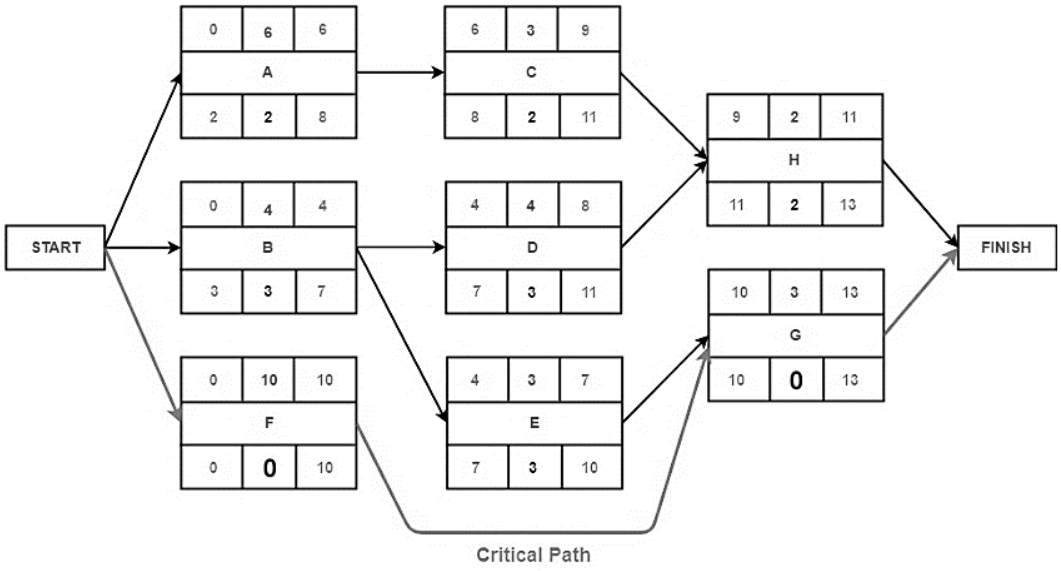


Fig.5.10 आयडेंटिफायिंग क्रिटिकल पाथ (Identifying Critical Path)

5.3.2 प्रोजेक्ट इव्हॅल्युएशन ॲण्ड रिव्ह्यू टेक्निक (PERT)

प्रोजेक्ट इवलुएशन अँड रेव्हिएव टेकनिक (PERT) ही एक प्रोसेस आहे ज्याद्वारे प्रोजेक्टमधील ॲक्टिव्हिटी त्यांच्या योग्य क्रमाने आणि वेळेनुसार दर्शवले जातात. हे एक शेड्यूलिंग टेकनिक आहे जे प्रोजेक्टमधील कार्यांचे वेळापत्रक तयार करण्यासाठी, कार्ये आयोजित करण्यासाठी आणि समाकलित करण्यासाठी वापरले जाते. PERT ही मुळात मॅनेजमेंट प्लॅनिंग आणि कंट्रोलसाठीची एक मेकॅनिझम आहे, जी एखाद्या विशिष्ट प्रोजेक्टसाठी एक ब्लूप्रिंट (आराखडा) प्रदान करते. प्रोजेक्टमधील सर्व प्राथमिक इव्हेंट्स किंवा घटक शेवटी PERT तंत्राद्वारे स्पष्टपणे ओळखले जातात. या टेकनिकमध्ये, एक PERT चार्ट तयार केला जातो, जो प्रोजेक्टमधील सर्व निर्दिष्ट टास्क्सचे आणि त्यांच्या वेळापत्रकाचे प्रतिनिधित्व करतो. PERT चार्टमधील टास्क्स किंवा इव्हेंट्सचे रिपोर्टिंग लेव्हल्स मुख्यतः वर्क ब्रेकडाउन स्ट्रक्चर (WBS) मध्ये परिभाषित केल्याप्रमाणेच असतात.

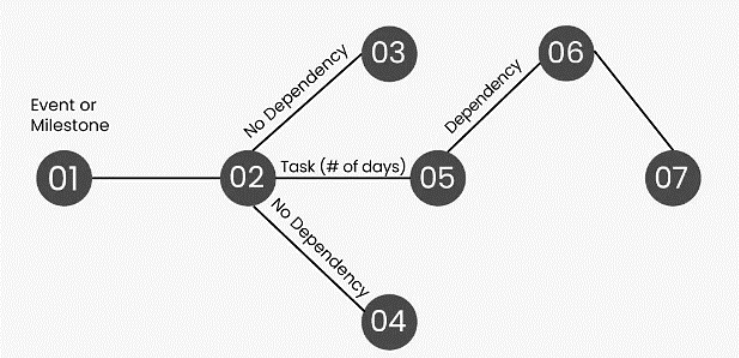


Fig. 5.11 पर्ट चार्ट (PERT Chart)

PERT चार्ट हे एक प्रोजेक्ट मॅनेजमेंट साधन आहे जे कार्यांचे नियोजन आणि वेळापत्रक तयार करण्यासाठी वापरले जाते. हे चार्ट प्रोजेक्टमधील क्रियाकलापांचा क्रम व त्यांचा कालावधी स्पष्टपणे दर्शवते. PERT चार्टचा वापर प्रोजेक्टमधील कार्यांचे वेळापत्रक तयार करण्यासाठी, त्यांचे आयोजन करण्यासाठी आणि समन्वय साधण्यासाठी केला जातो. PERT चार्टचे मुख्य उद्दीष्ट म्हणजे क्रिटिकल पाथ ओळखणे — ज्यामध्ये अशा क्रियाकलापांचा समावेश असतो जे वेळेत पूर्ण करणे आवश्यक असते. हे चार्ट प्रोजेक्ट नियोजन क्रियाकलापांमध्ये तयार केलेल्या माहितीच्या आधारे तयार केले जाते, जसे की — प्रयत्नांचा (effort) अंदाज, सॉफ्टवेअर विकासासाठी योग्य प्रोसेस मॉडेलची निवड, आणि कार्यांचे सबटास्कमध्ये विघटन.

5.3.2.1 PERT चार्टचे मुख्य घटक:

- **नोड्स (Nodes):** हे कार्य किंवा माइलस्टोन (महत्त्वाचे टप्पे) दर्शवतात. प्रत्येक नोडमध्ये कार्याचे नाव असते आणि त्या कार्याची कालावधी देखील दर्शविली जाऊ शकते.
- **बाण (Arrows):** हे कार्यांची दिशा, क्रम आणि त्यांच्या परस्पर अवलंबनाचे संकेत देतात. उदाहरणार्थ, जर A वरून B कडे बाण असेल, तर कार्य A पूर्ण झाल्यावरच कार्य B सुरू होऊ शकते.
- **वेळेचा अंदाज (Time Estimation):** कार्य पूर्ण करण्यासाठी लागणाऱ्या कालावधीचा अंदाज दिला जातो.
- **क्रिटिकल पाथ (Critical Path):** प्रोजेक्ट व्यवस्थापनातील हा सर्वात लांब कालावधीचा मार्ग आहे जो संपूर्ण प्रोजेक्ट पूर्ण होण्यासाठी आवश्यक असलेल्या किमान वेळेचा निर्धार करतो.
- **माइलस्टोन्स (Milestones):** हे प्रोजेक्ट टाईमलाइनमधील महत्त्वाचे टप्पे असतात, जे प्रगतीच्या दृष्टिकोनातून महत्त्वाच्या घटना किंवा डेडलाईन्स दर्शवतात.

5.3.2.2 PERT चार्ट ची कार्यप्रणाली (Working of PERT Chart):

PERT चार्टचा उपयोग प्रोजेक्टमधील कार्ये नियोजित करण्यासाठी आणि दृश्य स्वरूपात सादर करण्यासाठी केला जातो. तो संपूर्ण प्रोजेक्ट वेगवेगळ्या वैयक्तिक कार्यांमध्ये विभागतो आणि ती कार्ये कोणत्या क्रमाने पूर्ण केली जाणे आवश्यक आहे हे दर्शवतो. प्रत्येक कार्य नोडच्या स्वरूपात दर्शवले जाते, तर कार्यांमधील अवलंबन बाणांद्वारे दर्शवले जाते. या चार्टचे विश्लेषण करून, कार्यसंघ (team) क्रिटिकल पाथ ओळखू शकते, जो प्रोजेक्ट पूर्ण होण्यासाठी आवश्यक असलेला सर्वात

कमी कालावधी निश्चित करण्यात मदत करतो. यामुळे संसाधने (resources) अधिक प्रभावीपणे नियोजित व वितरित करता येतात.

5.3.2.2 .1 PERT चार्ट तयार करण्याचे टप्पे (Steps to Create a PERT Chart):

PERT चार्ट तयार करण्यासाठी, खाली दिलेल्या स्टेप्सचे अनुसरण करा:

स्टेप 1: प्रोजेक्ट ॲक्टिव्हिटीज ओळखा (Identify Project Tasks)

प्रोजेक्ट पूर्ण करण्यासाठी आवश्यक असलेल्या सर्व टास्क्स ची यादी करा. टास्क्स ओळखणे म्हणजे प्रोजेक्ट पूर्ण करण्यासाठी लागणाऱ्या सर्व चरणांची नोंद घेणे. सुरुवात मुख्य उद्दिष्ट (Goal) ओळखून करा आणि त्यानंतर त्याला छोटे-छोटे टास्क्स मध्ये विभाजित करा. Team सदस्यांकडून ideas घ्या आणि प्रत्येक टास्क साठी स्पष्ट ॲक्शन वर्डस वापरा.

स्टेप 2: टास्क्स अवलंबित्व (डिपेंडेंसिएस)

टास्क्स अवलंबित्व **(Dependencies) निश्चित करा (Define task dependencies)**

डिपेंडेंसिएस निश्चित करणे म्हणजे कोणते टास्क्स आधी पूर्ण करणे आवश्यक आहे हे ओळखणे. टास्क्स लिस्ट पाहा आणि विचार करा की एखादा टास्क्स दुसऱ्यावर अवलंबून आहे का. यामुळे कामाचा योग्य सीक्वेन्स तयार करता येतो.

स्टेप 3: टाईमलाइनचा अंदाज लावा (Estimate Timeline)

टाईमलाइन एस्टिमेट करणे म्हणजे प्रत्येक टास्क्स पूर्ण होण्यास किती वेळ लागेल याचा अंदाज लावणे. प्रत्येक टास्क्स साठी खालील वेळा विचारात घ्या: ऑप्टिमिस्टिक टाईम – काम लवकर पूर्ण झाल्यास लागणारा वेळ पेसिमिस्टिक टाईम – सर्वात जास्त वेळ लागल्यास मोस्ट लाईकली टाईम – सामान्यतः लागणारा वेळ हे प्रोजेक्ट ड्युरेशन समजून घेण्यास आणि प्लॅनिंग मध्ये मदत करते.

स्टेप 4: क्रिटिकल पाथ काढा (Calculate Critical Path)

क्रिटिकल पाथ काढणे म्हणजे अशा टास्क्सचा सगळ्यात लांब अनुक्रम शोधणे, ज्यावर प्रोजेक्टचा एकूण कालावधी अवलंबून असतो. हे असे टास्क्स असतात की जे विलंबित (डिले) झाले, तर पूर्ण प्रोजेक्टच विलंबित डिले होतो. त्यामुळे रिसोर्स अलोकेशन आणि प्लॅनिंग योग्य पद्धतीने करता येते.

स्टेप 5: टास्क प्रगती ट्रॅक करा (Manage task progress)

यामध्ये प्रत्येक कामाचे ट्रेकिंग कसे करतात याचा अंतर्भाव असतो प्रत्येक टास्कची प्रगती वेळोवेळी तपासा. टास्क्स शेड्युलप्रमाणे चालू आहेत का हे पहा. विलंब (डिले) झाल्यास त्यावर काम करा आणि गरज असल्यास प्लॅनिंगमध्ये बदल करा. यामुळे प्रोजेक्ट ट्रॅकवर राहतो आणि समस्या लवकर सोडवता येतात.

5.3.2.3 PERT चार्टची वैशिष्ट्ये (Characteristics of PERT Chart):

PERT ची मुख्य वैशिष्ट्ये खालीलप्रमाणे आहेत:

1. निर्णय घेण्याच्या अंमलबजावणीसाठी आवश्यक महत्त्वाची माहिती मिळवण्यासाठी हा एक आधार म्हणून वापरला जातो.
2. हा सर्व प्लॅनिंग क्रियाकलापांचा पाया असतो.
3. PERT व्यवस्थापनाला संसाधनांचा सर्वोत्तम वापर कसा करता येईल हे ठरवण्यास मदत करतो.
4. PERT मध्ये टाईम नेटवर्क ॲनालिसिस टेक्निक वापरून फायदे घेतले जातात.
5. PERT माहिती सादर करण्यासाठी एक ठराविक स्ट्रक्चर तयार करतो.
6. प्रोजेक्ट वेळेत पूर्ण करण्यासाठी आवश्यक असलेले महत्त्वाचे घटक ओळखण्यात व्यवस्थापनाला मदत करतो.
7. तो क्रिटिकल पाथमधील ॲक्टिव्हिटीज स्पष्टपणे दाखवतो.
8. निर्दिष्ट तारखेपूर्वी प्रोजेक्ट पूर्ण होण्याची संभाव्यता प्रॉबॅबिलिटी काय आहे, हे सांगतो.
9. एक किंवा अधिक टास्क्स एकमेकांवर कसे अवलंबून आहेत, याचे वर्णन करतो.
10. हा प्रोजेक्ट एका ग्राफिकल प्लॅन फॉर्ममध्ये सादर करतो.

5.3.2.4 PERT चार्टचे फायदे (Advantages of PERT Chart):

1. प्रोजेक्ट पूर्ण होण्यासाठी लागणाऱ्या वेळेचा एस्टिमेट PERT देतो.
2. स्लॅक टाईम असलेल्या ॲक्टिव्हिटीज ओळखण्यासाठी तो मदत करतो.

3. विशिष्ट प्रोजेक्टमधील ॲक्टिव्हिटीज कधी सुरू होतील आणि कधी पूर्ण होतील याच्या तारखा निश्चित करता येतात.
4. क्रिटिकल पाथमधील ॲक्टिव्हिटीज ओळखण्यासाठी तो प्रोजेक्ट मॅनेजरला सहाय्य करतो.
5. PERT मोठ्या प्रमाणातील डेटा सादर करण्यासाठी एक सुसंघटित डायग्राम तयार करतो.

5.3.2.5 PERT चार्टचे तोटे (Disadvantages of PERT Chart):

1. PERT अधिक गुंतागुंत (कॉम्प्लेक्सिटी) असल्यामुळे त्याची अंमलबजावणी करताना अडचणी येतात.
2. ॲक्टिव्हिटी टाईमचा एस्टिमेट व्यक्तिनिष्ठ असतो, जो PERT चा एक मोठा डिसॲडव्हान्टेज आहे.
3. PERT ची मेंटेनन्स ही महागडी आणि गुंतागुंतीची (कॉम्प्लेक्स) असते.
4. प्रत्यक्षातील डिस्ट्रिब्युशन, PERT च्या बीटा डिस्ट्रिब्युशनपेक्षा वेगळी असू शकते, ज्यामुळे चुकीची ॲसंप्शन्स तयार होतात.
5. PERT हे प्रोजेक्ट अपेक्षित वेळेपेक्षा कमी मध्ये होईल हे दाखवतो, कारण इतर काही पाथ्समधील ॲक्टिव्हिटीज जर विलंबित झाल्या, तर त्या पाथ्सही क्रिटिकल पाथ बनू शकतात.

टीप: प्रोजेक्ट इव्हॅल्युएशन आणि रिव्ह्यू टेक्निक (PERT) हे प्रभावी प्रोजेक्ट मॅनेजमेंटसाठी एक मौल्यवान साधन आहे. हे टीमला टास्क्सचा सीक्वेन्स समजून घेण्यास, त्यांच्या अवलंबित संबंधांचे आकलन करण्यास, आणि वेळेत प्रोजेक्ट पूर्ण होण्यासाठी क्रिटिकल पाथ ओळखण्यास मदत करते. PERT चा वापर करून संस्था प्लॅनिंग, रिसोर्स ॲलोकेशन, आणि एकंदर प्रोजेक्ट यशामध्ये सुधारणा करू शकतात, ज्यामुळे उद्दिष्टे कार्यक्षमतेने आणि परिणामकारकपणे पूर्ण करता येतात.

PERT आणि CPM मधील फरक

अ.क्र.	पैलू	PERT	CPM
1	पूर्ण रूप (Abbreviation)	PERT चे पूर्ण रूप Project Evaluation and Review Technique आहे.	CPM चे पूर्ण रूप Critical Path Method आहे.
2	परिभाषा (Definition)	ही एक प्रोजेक्ट मॅनेजमेंट तंत्र आहे जी अनिश्चित (uncertain) वेळ असलेल्या activities साठी वापरली जाते.	ही एक प्रोजेक्ट मॅनेजमेंट तंत्र आहे जी निश्चित (certain) वेळ असलेल्या activities साठी वापरली जाते.
3	स्वरूप (Orientation)	Event-oriented तंत्र आहे — network घटना (events) यांच्यावर आधारित तयार केला जातो.	Activity-oriented तंत्र आहे — network क्रियाकलापांवर (activities) आधारित तयार केला जातो.
4	मॉडेल प्रकार (Model Type)	हे एक probability model आहे.	हे एक deterministic model आहे.
5	लक्ष (Focus)	वेळेवर लक्ष केंद्रित करते — वेळेचे उद्दिष्ट गाठणे किंवा पूर्णतेचा टक्का मोजणे महत्त्वाचे असते.	वेळ आणि खर्च यामधील समतोलावर लक्ष केंद्रित करते — खर्च कमी करणे महत्त्वाचे असते.
6	अचूकता (Precision)	अतिशय अचूक वेळ मोजण्यासाठी योग्य.	साधारण वेळ मोजण्यासाठी योग्य.
7	कामाचे स्वरूप (Nature of Job)	नवीन व नॉन-रिपिटेटिव्ह (non-repetitive) स्वरूपाचे प्रोजेक्ट्ससाठी वापरले जाते.	पुनरावृत्ती होणाऱ्या (repetitive) प्रोजेक्ट्ससाठी वापरले जाते.

अ.क्र.	पैलू	PERT	CPM
8	क्रॅशिंग (Crashing)	क्रॅशिंग शक्य नाही, कारण वेळ निश्चित नसतो.	क्रॅशिंग शक्य आहे, कारण वेळ निश्चित असतो.
9	डमी ॲक्टिव्हिटी (Dummy Activities)	डमी ॲक्टिव्हिटी वापरत नाही.	डमी ॲक्टिव्हिटी वापरतो जेणेकरून activity चा क्रम नीट दाखवता येतो.
10	योग्यता (Sustainability)	संशोधन व विकास (R&D) प्रोजेक्टसाठी योग्य.	बांधकाम व पायाभूत सुविधा प्रोजेक्टसाठी योग्य.

References:

1. Software Engineering: A practitioner's approach by Roger S. Pressman & Bruce R. Maxim, McGraw Hill Higher Education, New Delhi, (Ninth Edition) ISBN 93-5532-504-5
2. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
3. Software Engineering: Principles and practices by Deepak Jain, Oxford University Press, New Delhi ISBN 9780195694840
4. Software Testing: Principles and Practices by Srinivasan Desikan, Gopalaswamy Ramesh, PEARSON Publisher: Pearson India 2007, ISBN: 978-81-7758-121-8
5. Software Testing by Ron Patton, Sams Publishing; 2nd edition, 2005 ISBN: 0672327988
6. Software Engineering Principles and Practice, Waman S. Jawadekar, McGraw Hill Education India 2011, ISBN-13: 978-0070583719

Websites:

1. https://www.sixsigmadaily.com/the-activity-network-diagram/
2. https://www.geeksforgeeks.org/software-engineering-critical-path-method/
3. https://www.projectmanager.com/guides/work-breakdown-structure

युनिट-6
सॉफ्टवेअर क्वालिटीअशुरन्स
(Software Quality Assurance)

विषय निष्पत्ती (Course Outcome):

CO 6: सॉफ्टवेअर डेव्हलपमेंट मध्ये गुणवत्ता हमी तत्त्वे वापरा.

घटक निष्पत्ती (Theory Learning Outcome - TLO):

1. सॉफ्टवेअर गुणवत्ता व्यवस्थापन आणिसॉफ्टवेअर गुणवत्ता आश्वासनयांच्यात फरक करणे.
2. सॉफ्टवेअर विकास प्रकल्पात सॉफ्टवेअर गुणवत्ता आश्वासनाचे टप्पेलागू करणे.
3. सॉफ्टवेअर गुणवत्ता मूल्यांकन मानके लागू करणे.

6.1 सॉफ्टवेअर गुणवत्ता व्यवस्थापन विरुद्ध सॉफ्टवेअर गुणवत्ता आश्वासन

6.1.1 सॉफ्टवेअर गुणवत्ता व्यवस्थापन

सॉफ्टवेअर गुणवत्ता व्यवस्थापन ही एक व्यापक फ्रेमवर्क आहे ज्या मध्ये सॉफ्टवेअर उत्पादनांच्या गुणवत्तेचे व्यवस्थापन आणि सुधारणा करण्याच्या उद्देशाने सर्व संघटनात्मक धोरणे, प्रोसेस आणि ॲक्टिव्हिटीज समाविष्ट आहेत. त्यात सॉफ्टवेअर विकासाचे संपूर्ण जीवनचक्र समाविष्ट आहे.

SQM चे मुख्य घटक:

1. **गुणवत्ता नियोजन:** ग्राहकांच्या अपेक्षा आणि व्यवसाय उद्दिष्टांशी सुसंगत गुणवत्ता मानके आणि उद्दिष्टे परिभाषित करणे. गुणवत्ता लक्ष्ये पूर्ण करण्यासाठी आवश्यक असलेल्या कार्यपद्धती, साधने आणि संसाधने स्थापित करणे.
2. **गुणवत्ता हमी (SQA):** प्रक्रियांचे पालन केले जाते आणि मानके पूर्ण केली जातात याची खात्री करण्यासाठी पद्धतशीर ॲक्टिव्हिटीज अंमलात आणा.
3. सॉफ्टवेअर उत्पादन निर्दिष्ट आवश्यकतापूर्ण करते हे सत्यापित करण्यासाठी वापरल्या जाणाऱ्या ऑपरेशनल तंत्रे आणि ॲक्टिव्हिटीज . चाचणी, तपासणी आणि पुनरावलोकने समाविष्ट आहेत.
4. **गुणवत्ता नियंत्रण (QC):** गुणवत्ता सुधारणा:प्रतिसाद, मेट्रिक्स आणि ऑडिटवर आधारित प्रोसेस, साधने आणि तंत्रे वाढविण्यासाठी सतत प्रयत्न.

SQM ची उद्दिष्टे (Objectives) :

1. ग्राहकांच्या अपेक्षा पूर्ण करणारी किंवा त्यापेक्षा जास्त सॉफ्टवेअर उत्पादने वितरित करणे.
2. दोष आणि पुनर्वापर खर्च कमी करणे.
3. सतत प्रोसेस सुधारणा स्थापित करणे.
4. व्यवसाय धोरणासह गुणवत्ता उद्दिष्टे संरेखित करणे.

साधने आणि तंत्रे (Tools & Techniques) :

1. सॉफ्टवेअर प्रोसेस मॉडेलिंग आणि सुधारणा फ्रेमवर्कजसे की CMMI (कॅपॅबिलिटी मॅच्युरिटी मॉडेल इंटिग्रेशन), ISO 9001.
2. मेट्रिक्स संकलन आणि विश्लेषण(दोषघनता, चाचणी कव्हरेज, अपयशाचा सरासरी वेळ).
3. जोखीम व्यवस्थापन आणि गुणवत्ता कमी करणे.

SDLC मधील भूमिका (Roles) :

SQM सॉफ्टवेअर डेव्हलपमेंट लाइफ सायकल (SDLC) च्या सर्व टप्प्यांचा समावेश करते, आवश्यकता गोळा करणे, डिझाइन,कोडिंग,चाचणी,तैनाती पासून देखभाली पर्यंत, गुणवत्ता व्यवस्थापन तत्त्वे संपूर्ण पणे अंतर्भूतआहेत याची खात्री करणे.

6.1.2 सॉफ्टवेअर गुणवत्ता आश्वासन (Software Quality Assurance) :

सॉफ्टवेअर गुणवत्ता आश्वासन हा SQM चा एक उपसंच आहे जो विकास प्रोसेसयोग्य रित्या पाळली जात आहे आणि सॉफ्टवेअर गुणवत्ता मानके आणि आवश्यकता पूर्ण करते याची खात्री करण्यासाठी पद्धतशीर ॲक्टिव्हिटीज आणि प्रक्रियांवर लक्षकेंद्रित करतो. ते प्रामुख्याने प्रोसेस-केंद्रितआहे.

SQA चे मुख्य उपक्रम (Core Activities of SQA) :

1. प्रोसेस व्याख्या आणि अंमल बजावणी:मानक कार्यपद्धती (SOPs), कोडिंग मानके, डिझाइन मानके आणि चाचणी मानके परिभाषित करणे.
2. ऑडिट आणि पुनरावलोकने :मानकांचे पालन तपासण्यासाठी विविध टप्प्यांवर औपचारिक तपासणी, वॉक थ्रू आणि ऑडिट करणे.
3. सत्यापन आणि प्रमाणीकरण: सत्यापन हे सुनिश्चित करते की उत्पादन योग्य रित्या तयार केले जात आहे (प्रक्रियेचे अनुसरण करून). प्रमाणीकरण योग्य उत्पादन तयार केले जात आहे याची खात्री करते (आवश्यकता पूर्ण करते).
4. चाचणी व्यवस्थापन:चाचणी क्रियाकलापांचे नियोजन आणि पर्यवेक्षण करणे (युनिट चाचणी, एकत्रीकरण चाचणी, सिस्टम चाचणी, स्वीकृती चाचणी).
5. दोष ट्रॅकिंग आणि अहवाल देणे दोषनोंदी राखणे, मूळ कारणे विश्लेषण करणे आणि दोष निराकरण सत्यापित करणे.

SQA चेउद्दिष्टे (Objectives) :

1. प्रक्रियांचे पालन सुनिश्चित करून दोषांना प्रतिबंधित करणे.
2. पुनरावलोकने आणि ऑडिटद्वारे लवकर दोष शोधणे.
3. सॉफ्टवेअर मानकांची पूर्तता करते असाविश्वास भागधारकांना.
4. ऑडिट निकालांवर आधारित सतत प्रोसेस सुधारणा सुलभ करणे.

साधने आणि तंत्रे (Tools & Techniques) :

1. ऑडिट आणि पुनराव लोकनांसाठी चेक लिस्ट.
2. स्वयंचलित चाचणी साधने (JUnit, Selenium).
3. कॉन्फिगरेशन व्यवस्थापन साधने (Git, SVN).
4. दोष ट्रॅकिंग सिस्टम (JIRA, Bugzilla).
5. प्रोसेस अनुपालन फ्रेमवर्क (ISO/IEC 12207, IEEE मानके).

SDLC मधील भूमिका (Roles) :

सॉफ्टवेअर प्रोसेस आणि उत्पादन गुणवत्ता देखरेख आणि सुधारण्यासाठी SQA ॲक्टिव्हिटीज संपूर्ण SDLC मध्ये आयोजित केले जातात, विशेषतः डिझाइन, अंमलबजावणी आणि चाचणी टप्प्यां दरम्यान.

फीचर	सॉफ्टवेअर क्वालिटी मॅनेजमेंट (SQM)	सॉफ्टवेअर क्वालिटी ॲश्युरन्स (SQA)
व्याप्ती	व्यापक	विशिष्ट
लक्ष केंद्रित	सामरिक	कार्यात्मक
समाविष्ट उपक्रम	नियोजनसुधारणा ,नियंत्रण ,	पुनरावलोकनेअनुपालन ,चाचणी ,
जबाबदारी	व्यवस्थापक आणि संघटनात्मकनेते	टीम QA आणि विकासक

6.2 सॉफ्टवेअर गुणवत्ता हमीचे टप्पे: नियोजन, उपक्रम, ऑडिट आणि पुनरावलोकन (Phases of Software Quality Assurance: Planning, activities, audit, and review)

6.2.1 नियोजन (Planning) :

अंतिम उत्पादन गुणवत्ता मानके आणि ग्राहकांच्या गरजा पूर्ण करते याची खात्री करण्यासाठी संपूर्ण सॉफ्टवेअर प्रकल्पातहा एक गुणवत्ता हमी टप्पा आहे.

प्रमुख घटक:

1. **गुणवत्ता हमी योजना (QAP):** SQA उद्दिष्टे, व्याप्ती, जबाबदाऱ्या, प्रोसेस, मानके आणि साधने यांचे वर्णन करणारे दस्तऐवज.
2. **संसाधन वाटप:** QA टीम सदस्य, साधने, प्रशिक्षण आणि आवश्यक बजेटओळखा.
3. **प्रक्रियेची व्याख्या:** अनुसरण करणेयच्या प्रोसेस आणि मानके निवडा आणि दस्तऐवजीकरण करणे (कोडिंग मानके, चाचणी प्रोटोकॉल, दस्तऐवजी करण मानके).
4. **वेळापत्रक:** प्रकल्पाच्या टप्प्यांशी जुळवून घेतलेल्या SQA क्रियाकलापांसाठी टाईमलाइन सेट करणे.
5. जोखीम ओळखणे: गुणवत्ता जोखीमांचा अंदाजघ्याआणि कमीकरण्याच्या धोरणांची व्याख्या करणे.

परिणाम (Outcome) :

एक औपचारिक योजना जी संपूर्ण SQA प्रक्रियेचे मार्गदर्शन करते, प्रत्येकाला त्यांच्या भूमिका आणि गुणवत्ता कशी व्यवस्थापित केली जाईल हे समजते याची खात्री करते.

6.2.2 उपक्रम (Activities)

दोष टाळण्यासाठी आणि गुणवत्ता मानकांचे पालन सुनिश्चित करण्यासाठी योजने नुसार सर्व SQA उपक्रमांची अंमलबजावणी वापर केली जाते.

प्रमुख उपक्रम:

1. **प्रोसेस देखरेख :** विकास, कोडिंग, चाचणीआणि तैनाती दरम्यान परिभाषित प्रक्रियांचे पालन ट्रॅक करणे.
2. **पुनरावलोकने आणि तपासणी :** दोष लवकर पकडण्यासाठी आवश्यकता,डिझाइन, कोड आणि चाचणी प्रकरणांचे समवयस्क पुनरावलोकने, वॉकथ्रू आणि औपचारिक तपासणी करणे.
3. **चाचणी :** सॉफ्टवेअरची कार्यक्षमता आणि कार्य प्रदर्शन सत्यापित करण्यासाठी विविधप्रकारच्या चाचणी (युनिट, एकत्रीकरण, प्रणाली, स्वीकृती) व्यवस्थापित करणे.
4. **कॉन्फिगरेशन व्यवस्थापन:** अखंडता आणि ट्रेसेबिलिटी राखण्यासाठी सॉफ्टवेअर आणि दस्तऐवजी करणातील बदल नियंत्रित करणे.
5. **दोष ट्रॅकिंग:** निराकरणे आणि बंद पडताळण्यासाठी दोष लॉग करणे, विश्लेषण करणे आणि अहवाल द्या.
6. **प्रशिक्षण आणि जागरूकता :** प्रकल्प सदस्यांना गुणवत्ता मानके आणि प्रक्रियांबद्दल शिक्षित करणे.

परिणाम:

SQA प्रक्रियांची सक्रिय अंमलबजावणी जी प्रोसेस शिस्त राखून, दोष लवकर शोधण्यास आणि दुरुस्तकरण्यास मदत करते.

6.2.3 ऑडिट

ऑडिटसाठी प्रक्रियांची प्रभावीता आणि मानके आणि SQA योजनेचे पालन यांचे औपचारिक मूल्यांकन करणे आवश्यक आहे. ऑडिट स्वतंत्र मूल्यांकन प्रदान करतात.

ऑडिटचे प्रकार:

- **प्रोसेस ऑडिट:** योजनेत परिभाषित केलेल्या प्रोसेस योग्यरित्या पाळल्या जात आहेत की नाही हे पडताळू न पाहणे.
- **उत्पादन ऑडिट:** सॉफ्टवेअर उत्पादन तपशील आणि गुणवत्ता आवश्यकता पूर्ण करते का ते तपासणे.
- **अनुपालन ऑडिट:** संस्थात्मक किंवा नियामकमानकांचे पालन मूल्यांकन करणे (उदा., ISO, CMMI).

प्रमुख ॲक्टिव्हिटीज:

- मानके आणि प्रक्रियांवर आधारित ऑडिटचे कलिस्ट तयार करणे.
- प्रकल्प कला कृतींची तपासणी करणे (कागदपत्रे, कोड, चाचणी अहवाल).
- प्रकल्प कर्मचाऱ्यांशी मुलाखती घेणे.
- गैर-अनुरूपता आणि सुधारणेसाठी क्षेत्रांसह ऑडिट निष्कर्षांचा अहवाल देणे.

निकाल:

अनुपालन स्थिती, विचलन, जोखीम आणि सुधारात्मक कृतींसाठी शिफारसी हायलाइट करणारा ऑडिट अहवाल देणे.

6.2.4 पुनरावलोकन

गुणवत्तेची उद्दिष्टे पूर्ण झाली आहेत किंवा नाही याची खात्री करण्यासाठी आणि सुधारण्याची संधी ओळखण्यासाठी प्रकल्पातील कामगिरी, प्रोसेसआणि एकूण प्रकल्प स्थितीचे मूल्यांकन करण्यासाठी पुनरावलोकने आवश्यक आहेत.

पुनरावलोकनांचे प्रकार:

1. व्यवस्थापन पुनरावलोकन: वरिष्ठ व्यवस्थापन धोरणात्मक निर्णय घेण्यासाठी प्रकल्प गुणवत्ता मेट्रिक्स, जोखीम आणि प्रगतीचे मूल्यांकन करते.
2. तांत्रिक पुनरावलोकन: तज्ञ आवश्यकता, डिझाइन किंवा कोड सारख्या डिलिव्हरेबलच्या तांत्रिक गुणवत्तेचे मूल्यांकन करतात.
3. समवयस्क पुनरावलोकन: टीम सदस्य एकमेकांच्या कामाचा अनौपचारिक किंवा औपचारिकपणे आढावा घेतात जेणे करून दोष लवकर शोधता येतील.
4. अंमलबजावणी नंतरचा आढावा: रिलीज झाल्यानंतर, प्रकल्पाचे यश, शिकलेले धडे आणि भविष्यातील सुधारणांसाठी क्षेत्रांचे मूल्यांकन करणे.

प्रमुख उपक्रम:

- गुणवत्ता डेटा आणि मेट्रिक्सचे संकलन आणि विश्लेषण करणे.
- निष्कर्ष आणि सुधारात्मक / प्रतिबंधात्मक कृतींची चर्चा करणे.
- पुनरावलोकन निकालांचे दस्तऐवजीकरण आणि कृती आयटम वर फॉलो-अपघेणे.

निकाल:

प्रोसेस आणि उत्पादनांमध्ये सतत सुधारणा, वाढलेली टीम जागरूकता आणि पुनरावृत्ती होणाऱ्या गुणवत्ता समस्यांना प्रतिबंध करणे.

6.3 गुणवत्ता मूल्यांकन मानके: सिक्स सिग्मा, सी एम एम आय: स्तर, प्रोसेस क्षेत्रे.

6.3.1 सिक्स सिग्मा:

सिक्स सिग्मा ही डेटा-चालित पद्धत आहे ज्याचा उद्देशदोष ओळखून तो दूर करून आणि परिवर्तनशीलता कमी करून प्रक्रियेची गुणवत्ता सुधारणे. 1980 च्या दशकात मोटोरोला ने सुरू केलेल्या सिक्स सिग्मा या सॉफ्टवेअर डेव्हलपमेंटच्या उद्योगांमध्ये एक लोक प्रिय गुणवत्तामानक बनला आहे. गुणवत्तेचा हा उच्च दर्जा एका संरचित सुधारणाचक्राचे पालन करून साध्य केला जातो,सामान्यतः डी एम ए आय सी प्रेम वर्क:परिभाषित करणे, मोजा, विश्लेषण करणे, सुधारणा करणे आणि नियंत्रण करणे. ही प्रोसेस संस्थांना समस्या स्पष्टपणे परिभाषित करण्यास, संबंधित डेटा गोळा करण्यास, मूळ कारणांचे विश्लेषण करण्यास, प्रभावी उपाय अंमलात आणण्यास आणि कालांतराने सुधारणा टिकवून ठेवण्यास मदत करते. सिक्ससिग्माचा वापर उत्पादन आणि आरोग्य सेवेपासून ते आयटी आणि वित्तपर्यंत, ऑपरेशन्स सुलभ करण्यासाठी, उद्योगांमध्ये केला जातो. वेस्ट कमी करणे आणि ग्राहकांचे समाधान वाढवा. परिमाणात्मक परिणाम आणि तथ्य आधारित निर्णय घेण्यावर त्याचे लक्षकेंद्रित केल्याने ते अशा वातावरणात विशेषतः मौल्यवान बनते जिथे सुसंगतता आणि कार्यक्षमता महत्त्वाची असते.

येथे 6 गोष्टी आहेत:

1. क्रमवारी लावा
    - o कामाच्या ठिकाणी अनावश्यक वस्तू काढून टाका.
    - o फक्त जे आवश्यक आहे तेच ठेवा.

2. व्यवस्थित करणे
    - o उर्वरित वस्तूता र्किक रित्या व्यवस्थित करणे.
    - o प्रत्येक गोष्टीला सहज प्रवेशासाठी एक नियुक्त जागा असल्याची खात्री करणे.
3. चमकवा
    - o कामाच्या ठिकाणी आणि उपकरणे नियमितपणे स्वच्छ करणे.
    - o नीटनेटके, सुरक्षित आणि आनंददायी कामाचे वातावरण राखा.
4. मानकीकरण करणे
    - o सुसंगत कार्यपद्धती आणि मानके स्थापित करणे.
    - o कामाच्या ठिकाणी सर्वोत्तम पद्धती सामान्य करणे.
5. टिकवून ठेवा
    - o सातत्याने मानकेराखा आणि पुनरावलोकन करणे.
    - o ६ S प्रणाली राखण्यासाठी सवयी आणि शिस्त निर्माण करणे.
6. सुरक्षितता
    - o कामाच्या ठिकाणी धोके ओळखा आणि दूर करणे.
    - o 6 S प्रक्रियेच्या प्रत्येक पैलू मध्ये सुरक्षितता समाकलित करणे.

सारांश: क्रमवारी लावा, व्यवस्थित करणे, चमकवा, मानकीकरण करणे, टिकवून ठेवा, सुरक्षितता.

6.3.1.1 सिक्स सिग्माची मुख्य तत्वे

1. **ग्राहकांचे लक्ष:** प्रत्येक गोष्ट ग्राहकांच्याअपेक्षा पूर्णकरण्यासाठी किंवा त्यापेक्षा जास्त करण्यासाठी सज्ज आहे.
2. **डेटा -चालित निर्णय घेणे :**सुधारणांचे मार्गदर्शन करण्यासाठी सांख्यिकीय पद्धती आणि मेट्रिक्स वापरते.
3. **प्रोसेस लक्ष :**प्रोसेस नियंत्रित करून सुधारणा येते व गुणवत्ता येते असे समजते.
4. **सक्रिय व्यवस्थापन** :वस्तू नंतर प्रतिक्रिया देण्याऐवजी समस्या उद्भवण्यापूर्वीच त्या टाळा.
5. सहकार्य गुणवत्तेच्या समस्या सोडवण्यासाठी एकत्रितपणे काम करणाऱ्या क्रॉस-फंक्शनल टीम चा समावेश आहे.
6. सतत सुधारणा नेहमी गुणवत्ता आणि कार्य क्षमता वाढवण्याचे मार्ग शोधणे.

6.3.1.2 सिक्स सिग्मा पद्धती

दोन मुख्य सिक्स सिग्मा प्रकल्प पद्धती आहेत, दोन्ही प्रोसेस गुणवत्ता सुधारण्या भोवती रचलेल्या आहेत:

1. डी एम ए आय सी (विद्यमान प्रोसेस सुधारण्यासाठी)

- **परिभाषित करणे:**
  समस्या, प्रकल्प उद्दिष्टे आणि ग्राहकांच्या आवश्यकता परिभाषित करणे.
  उदाहरण: नवीन तमरिलीझ मध्ये सॉफ्टवेअर बग कमी करणे.
- **मोज माप:**
  सध्याच्या प्रोसेस आणि दोषां वर डेटा गोळा करणे.
  उदाहरण: प्रत्येक मॉड्यूल मध्ये नोंदवलेल्या बगची संख्या ट्रॅक करणे.
- **विश्लेषण करणे:**
  सांख्यिकीय विश्लेषण वापरून दोष आणि प्रोसेस कार्यक्षमतेची मूळ कारणे ओळखणे.
  उदाहरण: बग पॅटर्न, कोडजटिलता किंवा विकासक पद्धतींचे विश्लेषण करणे.
- **सुधारणा करणे:**
  मूळ कारणांना संबोधित करण्यासाठी उपाय विकसित करणे आणि अंमलात आणा.
  उदाहरण: कोडपुनराव लोक ने, स्वयंचलित चाचणी किंवा रीफॅक्टरिंग सादर करणे.
- **नियंत्रण:**
  नफा टिकवून ठेवण्यासाठी आणि प्रतिगमन रोखण्यासाठी सुधारित प्रक्रिये चे निरीक्षण करणे.
  उदाहरण: रिलीझ नंतर बगसंख्या ट्रॅक करणारे डॅश बोर्ड सेट करणे.

2. डी एम ए डी व्ही (नवीन प्रोसेस / उत्पादने तयार करण्यासाठी किंवा पुनर्रचना करण्या साठी)

- **परिभाषित करणे:** ग्राहकांच्या गरजांशी जुळणारे प्रकल्प उद्दिष्टे परिभाषित करणे.
- **मोजमाप:** ग्राहकांच्या गरजा व तपशील मोजा आणि निश्चित करा.
- **विश्लेषण करणे:** ग्राहकांच्या गरजा पूर्ण करण्यासाठी प्रोसेस पर्यायांचे विश्लेषण करणे.
- **डिझाइन:** नवीन प्रोसेस किंवा उत्पादन डिझाइन करणे.
- **पडताळणी करणे:** डिझाइनची कार्यक्षमता आणि आवश्यकता पूर्ण करण्याची क्षमता सत्यापित करणे.

6.3.1.3 सिक्स सिग्मा मधील भूमिका

- **कार्यकारी नेतृत्व:** प्रायोजक आणि दर्जेदार उपक्रमांचे विजेते.
- **विजेते:** व्यवसाय युनिट्स मध्ये सिक्स सिग्मा प्रकल्पांचे व्यवस्थापन आणि प्रचार.
- **मास्टर ब्लॅक बेल्ट्स:** सिक्स सिग्मा पद्धती आणि साधनांवर तज्ञ प्रशिक्षक आणि मार्गदर्शक.
- **ब्लॅक बेल्ट्स:** समस्या सोडवण्याचे प्रकल्पांचे नेतृत्व करणे, सिक्स सिग्मा तंत्रे पूर्ण वेळ लागू करणे.
- **ग्रीन बेल्ट्स:** नियमित कर्तव्ये पार पाडताना अर्ध वेळ प्रकल्पांना समर्थन द्या.
- **टीम सदस्य:** प्रकल्पांमध्ये सहभागी व्हा, प्रोसेस ज्ञान प्रदान करणे.

6.3.1.4 सिक्स सिग्मामधील प्रमुख साधने आणि तंत्रे

- **सांख्यिकीय विश्लेषण:** गृहीतक चाचणी, प्रतिगमन विश्लेषण, ANOVA, नियंत्रण चार्ट.
- **प्रोसेस मॅपिंग:** फ्लोचार्ट, SIPOC (पुरवठादार, इनपुट, प्रोसेस, आउटपुट, ग्राहक) आकृत्या.
- **मूळ कारण विश्लेषण:** फिशबोन आकृत्या (इशिकावा), 5 व्हेस तंत्र.
- **अपयश मोड आणि परिणाम विश्लेषण (FMEA):** संभाव्य अपयश बिंदू ओळखणे आणि प्राधान्य देणे.
- **प्रयोगांची रचना (DOE):** प्रोसेस ऑप्टिमाइझ करण्यासाठी अनेकचलांची चाचणी करणे.
- **नियंत्रण चार्ट:** कालांतराने प्रक्रियेच्या वर्तनाचे निरीक्षण करणे.

6.3.1.5 सॉफ्टवेअर गुणवत्तेत सिक्ससिग्मा लागू करणे

- **सॉफ्टवेअर दोष कमी करणे:** बग किंवा अपयशाची कारणे ओळखण्यासाठी आणि दूरकरण्यासाठी DMAIC वापरणे.
- **चाचणी कार्यक्षमता सुधारणे:** चाचणी कव्हरेज आणि दोष शोधण्याचे दर विश्लेषण आणि ऑप्टिमायझ करणे.
- **विकास प्रोसेस सुव्यवस्थित करणे:** प्रोसेस सुधारणांद्वारे सायकल वेळ आणि (Rework)पुनर्कामकमी करणे.
- **ग्राहक समाधान वाढवणे:** सिक्स सिग्मा मेट्रिक्स वापरून ग्राहकांच्या गरजांनुसार सॉफ्टवेअर वैशिष्ट्ये आणि कार्यप्रदर्शन संरेखित करणे.

6.3.2 सी एम एम आय

सी एम एम आय ही एक प्रोसेस सुधारणा चौकट आहे जी संस्थांनात्यांच्या प्रोसेस विकसित करण्यास आणि परिष्कृत करण्यास मदत करते जेणे करून कार्यक्षमता, उत्पादन गुणवत्ता आणि अंदाज सुधारेल. सॉफ्टवेअर डेव्हलपमेंट, सिस्टम अभियांत्रिकी आणि इतर अभियांत्रिकी शाखां मध्ये याचा मोठ्या प्रमाणात वापर केला जातो.

6.3.2.1 सी एम एम आयस्तर

सी एम एम आय 5 परिपक्वता स्तर परिभाषित करते जे संस्थेच्या प्रक्रियांच्या तात्पुरत्या आणि अराजकते पासून परिपक्व आणि सतत सुधारण्या पर्यंतच्या उत्क्रांतीचे प्रतिनिधित्व करतात.

स्तर 1: प्रारंभिक

- प्रोसेस अप्रत्याशित, खराब नियंत्रित आणि प्रतिक्रिया शील असतात.

स्तर 2: व्यवस्थापित प्रोसेस क्षेत्रे

- **आवश्यकता व्यवस्थापन (REQM):** आवश्यकता व्यवस्थापित करणे आणि प्रकल्प योजनां नुसार संरेखन सुनिश्चित करणे.
- **प्रकल्प नियोजन (PP):** प्रकल्प ॲक्टिव्हिटीज परिभाषित करणाऱ्या योजना स्थापित करणे आणि राखणे.

- **प्रकल्प देखरेख आणि नियंत्रण (PMC):** प्रकल्प प्रगतीचा मागोवा घेणे आणि सुधारात्मक कृती करणे.
- **पुरवठा दारकरणेर व्यवस्थापन (SAM):** पुरवठादारांसोबत करणे व्यवस्थापित करणे.
- **मापन आणि विश्लेषण (MA):** व्यवस्थापन माहितीच्या गरजांना समर्थन देण्यासाठी मापन क्षमता विकसित करणे.
- **प्रोसेस आणि उत्पादन गुणवत्ता आश्वासन (PPQA):** प्रोसेस आणि उत्पादनांचे वस्तुनिष्ठ मूल्यांकन प्रदान करणे.
- **कॉन्फिगरेशन मॅनेजमेंट (CM):** कामाच्या उत्पादनांची अखंडता स्थापित करणे आणि राखणे.

स्तर 3: परिभाषित प्रोसेस क्षेत्रे

- **आवश्यकता विकास (RD):** ग्राहकांच्या गरजा ओळखणे, विश्लेषण करणे आणि विकसित करणे.
- **तांत्रिक उपाय (TS):** उपायांची रचना करणे, विकसित करणे आणि अंमलबजावणी करणे.
- **उत्पादन एकत्रीकरण (PI):** उत्पादन घटक एकत्र करणे आणि एकत्रीकरण सत्यापित करणे.
- **पडताळणी (VER):** उत्पादने आवश्यकतापूर्ण करतात या ची खात्री करणे.
- **प्रमाणीकरण (VAL):** उत्पादने त्यांच्या ऑपरेशनल वातावरणात इच्छित वापर पूर्ण करतात याची खात्री करणे.
- **संघटनात्मक प्रोसेस फोकस (OPF):** संघटनात्मक प्रोसेस सुधारणांची योजना करणे आणि तैनात करणे
- **संघटनात्मक प्रोसेस व्याख्या (OPD):** संघटनात्मक प्रोसेस मालमत्ता विकसित करणे आणि राखा.
- **संघटनात्मक प्रशिक्षण (OT):** लोकांचे कौशल्य आणि ज्ञान विकसित करणे.
- **एकात्मिक प्रकल्प व्यवस्थापन (IPM):** संघटनात्मक प्रोसेस वापरून प्रकल्प व्यवस्थापित करणे.
- **जोखीम व्यवस्थापन (RSKM):** प्रकल्प जोखीम ओळखणे आणि कमी करणे.
- **निर्णय विश्लेषण आणि निराकरण (DAR):** पर्यायांचे मूल्यांकन करण्यासाठी पद्धतशीर पद्धती वापरा.

स्तर 4: परिमाणात्मक रित्या व्यवस्थापित प्रोसेस क्षेत्रे

- **संघटनात्मक प्रोसेस कामगिरी (OPP):** प्रोसेस कामगिरी बेसलाइन स्थापितकरणे आणि राखा.
- **परिमाणात्मक प्रकल्प व्यवस्थापन (QPM):** सांख्यिकीय आणि इतर परिमाणात्मक तंत्रांचा वापर करून प्रकल्पांचे परिमाणात्मक व्यवस्थापन करणे.

स्तर 5: प्रोसेस क्षेत्रांचे ऑप्टिमायझेशन

- **संघटनात्मक नवोपक्रम आणि तैनाती (OID):** नवीन प्रोसेस सुधारणा आणि नवोपक्रम लागू करणे.
- **कारणात्मक विश्लेषण आणि निराकरण (CAR):** दोष आणि समस्यांची कारणे ओळखा आणि प्रतिबंधात्मक कृती करणे.

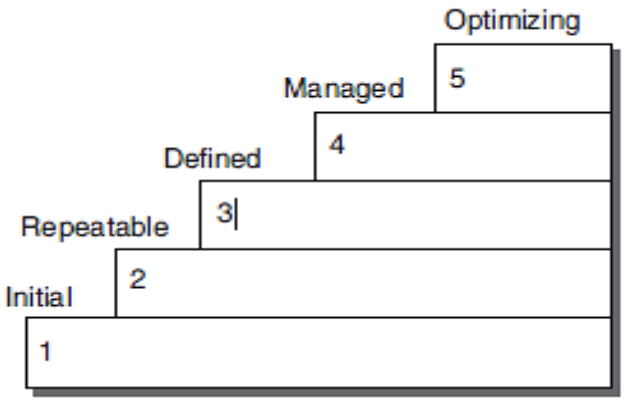


Fig 6.1: सीएमएमआय मॅच्युरिटी लेव्हल (CMMI Maturity Levels)

6.3.2.2 प्रोसेस क्षेत्रे

प्रोसेसक्षेत्र (पीए)	उद्देश
आवश्यकता व्यवस्थापन (REQM)	प्रकल्पाच्या संपूर्ण जीवनचक्रात आवश्यकता व्यवस्थापित करणेआणि नियंत्रित करणे.
प्रकल्प नियोजन (PP)	कार्ये, संसाधने आणि वेळापत्रक परिभाषित करणारे प्रकल्प योजना तयार करणेआणि देखरेख करणे.
प्रकल्प देखरेख आणि नियंत्रण (PMC)	प्रकल्पाच्या प्रगतीचे निरीक्षण करणे आणि आवश्यकतेनुसार सुधारात्मक कृती करणे.
मापन आणि विश्लेषण (MA)	प्रकल्प आणि व्यवसाय उद्दिष्टांना समर्थन देण्यासाठी मोजमाप क्षमता विकसित करणे आणि टिकवून ठेवणे.
प्रोसेस आणि उत्पादन गुणवत्ता हमी (PPQA)	मानकांचे पालन सुनिश्चित करण्यासाठी प्रोसेस आणि उत्पादनांचे वस्तुनिष्ठ मूल्यांकन करणे.
कॉन्फिगरेशन मॅनेजमेंट (सीएम)	प्रकल्प कार्य उत्पादनांमध्ये बदल व्यवस्थापित करणे आणि अखंडता राखणे.
पुरवठादार करणेर मॅनेजमेंट (एसएएम)	उत्पादने किंवा सेवांच्या बाह्य पुरवठादारांशी संबंध व्यवस्थापित करणे.

References:

1. Software Engineering: A practitioner's approach by Roger S. Pressman & Bruce R. Maxim, McGraw Hill Higher Education, New Delhi, (Ninth Edition) ISBN 93-5532-504-5
2. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
3. Software Engineering Concepts by Richard Fairly, McGraw Hill Education New Delhi -2001, ISBN-13: 9780074631218
4. International standards (e.g., ISO 9001, IEEE 730, CMMI documentation)
5. Software Quality Assurance by Daniel Galin

Websites:

1. https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/
2. https://www.tutorialspoint.com/software_engineering/index.htm
3. IEEE 730 – Software Quality Assurance Standard - https://standards.ieee.org/
4. ISO 9001 for Software Quality - https://www.iso.org/standard/62085.html
5. ISACA / CMMI – https://www.isaca.org/enterprise/cmmi